RAIRO-Oper. Res. 45 (2011) 241–294
DOI: 10.1051/ro/2011114

RAIRO Operations Research
www.rairo-ro.org

A SURVEY ON COMBINATORIAL OPTIMIZATION IN DYNAMIC ENVIRONMENTS*

Nicolas Boria[1] and Vangelis T. Paschos[1,2]

Abstract. This survey presents major results and issues related to the study of NPO problems in dynamic environments, that is, in settings where instances are allowed to undergo some modifications over time. In particular, the survey focuses on two complementary frameworks. The first one is the reoptimization framework, where an instance I that is already solved undergoes some local perturbation. The goal is then to make use of the information provided by the initial solution to compute a new solution. The second framework is probabilistic optimization, where the instance to optimize is not fully known at the time when a solution is to be proposed, but results from a determined Bernoulli process. Then, the goal is to compute a solution with optimal expected value.

Keywords. Approximation, reoptimization, hereditary problem, complexity, graph, on-line algorithms, probabilistic optimization.

Mathematics Subject Classification. 05C22, 05C69, 05C85, 68Q10, 68Q17, 68Q25, 68Q87, 68W25, 68W27.

1. Introduction

It is commonly known that many optimization problems are intractable, so that no efficient (*i.e.*, polynomial) optimal algorithms are known for them. Although it has not been clearly proved that such algorithms do not exist, it is commonly

Received October 31, 2011. Accepted November 14, 2011.

* *Research supported by the French Agency for Research (ANR) under the DEFIS program TODO, ANR-09-EMER-010.*

[1] LAMSADE, CNRS UMR 7243 and Université Paris-Dauphine, Place du Maréchal de Lattre de Tassigny, 75775 Paris Cedex 16, France. paschos@lamsade.dauphine.fr

[2] Institut Universitaire de France.

believed and accepted that it is the case, and the whole complexity theory relies today on this assumption.

The main approach to tackle these NP-hard problems consists in formulating approximation algorithms which run in polynomial time, and whose quality (measured as the so-called approximation ratio, *i.e.*, the ratio between the approximate and the optimal solutions), is somehow guaranteed. This approach has been widely applied and analyzed for the past twenty years, leading to the development of many approximation algorithms, as well as a whole classification of problems based on their approximability.

At the same time, some new paradigms were investigated, mainly for practical applications, where problems are often neither static nor perfectly known in advance. These uncertain or dynamic aspects of the problems that need to be solved can be modeled and dealt with in various ways. This survey focuses on two different and complementary kinds of dynamic models: reoptimization (and its generalization, *i.e.*, dynamic optimization) and probabilistic combinatorial optimization.

2. Preliminaries

In what follows, we first present in this an overview of the results achieved in dynamic optimization with emphasis to the reoptimization paradigm, as well as main techniques used in reoptimization algorithms. In the early 1980's new computational frameworks were applied to some polynomial problems such as MIN SPANNING TREE [58] or SHORTEST PATH [56,113], where the goal was to maintain an optimal solution under some more or less slight instance perturbations such as edge insertions, deletions or weight modifications. These innovative approaches were soon followed by many works improving on their results, mainly focusing on the data structures allowing complexity improvements in maintaining optimal solutions. Most of these results were dealing with polynomial problems, and aimed at maintaining optimal solutions in fully dynamic situations. More precisely, the paradigm was a little different from what is known today as reoptimization, considering that the instances could be subject to many consecutive or simultaneous perturbations, and the data structures aimed at computing new optima on perturbed instances as fast as possible, in particular, faster than an *ex nihilo* computation. Rather than *reoptimization* algorithms, the approach aimed at defining what was called *fully dynamic* algorithms. Indeed, the results in this field concerned data structures rather than algorithms. Later on, some NP-hard problems such as BIN PACKING were also analyzed in a full dynamic setting, where the goal is to maintain the bound on the approximation ratio.

It is only in the early 2000's that the concept of reoptimization as it is known today emerges [3]. This new paradigm considers a two-steps (initial and perturbed) environment instead of a fully dynamic one, and the model clearly meets various practical situations, especially when the problem to solve cannot be completely known in advance, and the time window between the moment when the full problem is revealed and the moment where the solution is needed is so short that an optimal "static" computation is impossible. In this case, which occurs very often in practical applications, it seems reasonable to devote a tremendous time to compute an optimal solution on the part of the instance which is known for sure, and then quickly adapt this solution to the full instance once it is revealed. Reoptimization algorithms consist in these quick adaptation methods.

In this new framework, many new problems and types of perturbations can be analyzed and solved. In particular, the application of the reoptimization paradigm to NP-hard problems has been intensively studied in the past few years. These studies involve vertex-set perturbation also, whereas before the introduction of this paradigm, only edge-set perturbation where handled. In many cases, maintaining an optimal solution in polynomial time is proved to be impossible unless P=NP. Indeed, a polynomial optimal reoptimization strategy `A` could end up solving any instance I in polynomial time, by starting with a trivial instance (for example an empty graph), and building incrementally the instance I with a polynomial number of steps. Applying the algorithm `A` at each step, one comes up with an optimal solution on I computed in polynomial time, which is impossible for NP-hard problems. On the other hand, it is straightforward to affirm that a reoptimization problem should be at least as easy as its static version, since one can always use a static algorithm to solve the reoptimization problem from scratch, as if no information on the initial optimum were available.

Finally, it is quite clear that, regarding the classical P *vs.* NP dichotomy, a given reoptimization problem belongs to the same complexity class as its static version. But this "complexity heredity" between a static optimization problem and its reoptimization version goes no further. In particular, one cannot extend this result to approximability complexity classes in NP (for example a problem hard to approximate in its static version can be APX in the reoptimization setting), so that the research in this field mainly – and successfully – focused on the definition of approximation algorithms. Indeed, reoptimizing NP-hard problems can lead to two kinds of advances:

- either the reoptimization algorithm achieves the same solution quality (optimality or approximation ratio) as the best "static" algorithm known, but with a better running time;
- or the reoptimization algorithm achieves in polynomial time a better approximation than the best approximation ratio known for the static version.

In practice, most results achieved in the reoptimization setting consist in approximation algorithms which guarantee better approximation ratios than the static algorithms, or even approximation ratios that *cannot* be guaranteed in the static case unless P=NP.

The most spectacular example is probably the classical MAX INDEPENDENT SET problem, for which it has been proved that no algorithm can achieve an approximation ratio better than $n^{\epsilon-1}$ (*i.e.*, asymptotically returning a trivial solution consisting of 1 vertex), unless P=NP. In the reoptimization setting (considering node insertions or deletions), the problem becomes APX in its weighted version, and even PTAS in its unweighted version. This setting has also been successfully applied to various classical NP-hard problems such as several scheduling problems [14, 15, 115], Steiner tree [28, 35, 36, 55] or the travelling Salesman problem [3,8,9,32,34]. In what follows we will discuss some general issues and properties concerning reoptimization of NP-hard optimization problems and we will review some of the most interesting applications.

An alternative way of handling dynamic data, quite complementary to reoptimization, is the probabilistic optimization paradigm, and discuss the main issues related to it. This paradigm includes a wide range of optimization problems, whose common characteristic is the explicit consideration of probabilities within the problem definitions. Thus, these problems are referred to as probabilistic combinatorial optimization problems (PCOPs).

On a very general level, PCOPs can be described as follows: one wishes to optimize a given problem Π, on an instance that cannot be known precisely in advance. Instead of having a fixed instance I of the problem Π to optimize, one has a whole set Ω of possible instances, each associated with a given probability. How one deals with such situations depends mainly on the probabilistic model considered.

Basically, probabilities can be introduced in any optimization model at two different levels:

- one or several parameters of the model can be subject to probabilities. For example, the weight of an edge can follow a uniform distribution $U(a, b)$, instead of being fixed as it is in classical optimization models. In this case, the structure of the problem is fixed, and some parameters within this structure are allowed to vary following a given probability distribution;
- the presence of one or several elements in the model can be subject to probabilities. For example, a node can be present in a graph, or a constraint can be present in a linear program, with a given probability. In this case, it is the structure of the problem itself which cannot be known in advance, and it eventually results from several Bernoulli trials.

Research in this field first focused on the former kind of model, while it now also focuses on the latter. As described in [26], the history of probabilistic analysis of combinatorial optimization models can be divided in three main phases:

The first models of combinatorial optimization problems including probabilities date back from the late 50's, and focused also on *probabilistic analysis in the Euclidean plane*. Precisely, the main results consisted in general properties of optima, in various problems where nodes were randomly placed in an euclidean plane. The first result concerned the TSP [16], and consisted in an analysis of optimal tours in an euclidean plane, with randomly placed nodes. This result was used as basis for an approximation algorithm for the same problem in [90], and many years later, was generalized in [120], and extended to a broad class of combinatorial optimization problems, including SHORTEST PATH, RECTILINEAR STEINER TREE, and MINIMUM MATCHING.

It is only in the late 70's that probabilistic analyses were led on non-Euclidean combinatorial optimization models. Namely, the results focused on *probabilistic analysis on problems with random lengths*. In this framework, two main problems were analyzed: NON SYMMETRIC TSP [91], and MIN SPANNING TREE [60, 121].

Notice that these two first phases include probabilities to randomize some parameters of the models (*i.e.*, position of nodes in the plane, or distances between nodes), and achieve limit theorems about optimal values. In other words, the goal was not to solve combinatorial optimization problems including random variables, but rather to perform probabilistic analyses of these problems, leading to general limit properties. For example, the value of an optimal solution of a given probabilistic problem can be shown to tend to a given function $f(n)$ with probability 1 when n tends to the infinity. Moreover, the set of elements (*i.e.*, nodes) were fixed, and only the relations between these elements (*i.e.*, the lengths of edges or arcs) were considered random.

Regarding these two last remarks, the third - and still ongoing - phase, *probabilistic combinatorial optimization*, proposes a coherent theoretical framework and, in many cases, seems to meet practical needs much better than "deterministic" optimization. It was initiated with the grounding results of Jaillet and Bertsimas. Jaillet [80, 81] introduced the PROBABILISTIC TRAVELING SALESMAN (PTSP), and a few years later, Bertsimas [21] analyzed many other PCOP's, and defined the concept of *a priori optimization*. Since then, many other classical optimization problems have been tackled in this probabilistic framework, and many results were provided, some of which will be reviewed in what follows. Basically PCOP's consist in adding a vector of probabilities associated with nodes (when dealing with a graph problem), that represent the probability for each node to be present in the instance that will actually need to be optimized. This instance thus results from n independent Bernoulli trials.

There are two major motivations for introducing elements subjects to probabilities in combinatorial optimization problems. The main motivation is, of course, the capacity to model real-world situations, where problem data are very often not as clearly defined as in deterministic optimization models. In many situations, randomness represents a major characteristic of the problems that need to be solved, that is why it seems natural to investigate optimization problems which take this randomness into account. There is a wide range of important and interesting applications of PCOPs, especially in the context of distribution planning, communication and transportation networks, job scheduling, organizational structures, etc. For such applications, the probabilistic nature of PCOPs makes them particularly attractive as models of real-world systems. The second motivation is related to robustness analysis. Indeed, consider a solution S on a given instance of a problem, including probabilities. A probabilistic analysis of S will provide information on how S "behaves" in the defined probabilistic space, and thus, somehow on its robustness.

The paper is organized as follows: in Section 3 which is devoted to describing "fully dynamic" algorithms and corresponding data structures, major results are presented, for both P and NP-hard problems. Section 4 deals with general properties of reoptimization. In Section 5, the computational efficiency of reoptimization applied to various NP-hard problems is discussed, and the latest results of this ongoing research field are presented, whereas Section 5.5 is specifically devoted to vehicle-routing problems, considering the huge flow of literature on the matter. In Section 6, devoted to probabilistic combinatorial optimization, rather than reviewing all the results achieved in this framework, we will discuss in detail two basic notions of it, namely, those of "probabilistic combinatorial optimization problem" and of "*a priori* optimization" and we will present the major issues related to them. Note that we will only discuss problems that can be interpreted in terms of graphs, although other kinds of problems have been studied using this approach, such as scheduling problems [44] or BIN PACKING [18, 76].

3. Fully dynamic algorithms

As explained before, the first problems studied in a dynamic setting – where instances of optimization problems are considered to be modified locally, and solutions are adapted given these input modifications –, were rather easy (*e.g.* polynomial) problems, for which this additional feature can be considered as an opportunity to reduce the computational cost. As stated before, a static algorithm can always be used to solve a reoptimization problem, so that the reoptimization version of a given optimization problem is always at least as easy as its static version. In particular, if a static problem Π is polynomial, its reoptimization (and fully dynamic) version $R(\Pi)$ is also polynomial, so that the only kind of result one can expect is a computational cost reduction.

Later on, the concepts of fully- and semi-dynamic graphs and algorithms were also applied to NP-hard problems, where the goal is to maintain a bounded competitive ratio (the ratio between the optimal solution, and the solution computed iteratively, along with each graph update) regardless of the nature or number of graph updates that might occur. In this case, the dynamic feature of graphs is clearly a generalization of the "on-line" model, and is considered as a challenge rather than as an opportunity, in the sense that designing dynamic algorithms which can maintain good solutions under any kind of graph perturbations is quite a hard task, somehow harder to compute good solutions from scratch.

Indeed, the main difference between polynomial, and NP-hard problems regarding the fully dynamic setting, is that, for the polynomial case, optimality is maintained after each update, and the optimality at a given stage helps computing an optimal solution for the next stage in a more efficient way than computing this solution from scratch. In the opposite, when one has to handle an NP-hard problem in a fully dynamic setting, one can only get an approximate solution at all time, and it is hard to ensure that the competitive ratio remains bounded from one stage to the other.

To our knowledge, only three NP-hard problems were analyzed in fully dynamic environment: BIN PACKING [79], MIN VERTEX COVER [78], and SHORTEST PATH IN PLANAR GRAPHS [93].

3.1. *Update* and *query* procedures

An algorithm for a given graph problem is said to be *dynamic* if it can maintain a solution to the problem as the graph undergoes changes. Regarding the polynomial problems analyzed in this setting, the changes considered are insertions or deletions of edges, or changes in the weight of some edges (if relevant). At each step, a more or less complex data structure is updated, and this data structure can be used at any time as input to compute a solution in a more efficient way than considering the raw instance as input. The data structure is like a pre-processed version of the instance, which one can use to compute optimal solutions faster. Of course, the process is of interest only when the data structure is easier to update (*i.e.*, requires a smaller number of operations to be updated) than the optimum itself.

In fully dynamic settings, an *update* operation denotes a change in the data structure in response to an incremental change to the input, and a *query* is a request for some information about the current solution, using the underlying data structure. These two notions are fundamental in the analysis of dynamic algorithms, since the complexity improvement relies completely on them.

Indeed, dynamic algorithms consist in maintaining a specific data structure at each step, rather than computing an optimal solution at each step. Whenever one wishes to have an optimal solution, one can run the query procedure. Very often, a trade-off has to be found between a structure easily updated, and one easily queried. This is precisely why complexities of both *update* and *query* operations are always analyzed.

Consider, for example, that one wants to maintain a minimum spanning tree in a dynamic graph using the classical Kruskal's algorithm [95]. To compute it from scratch, one needs to:

- sort all edges in increasing weight order (which takes $O(m\log(m))$ operations);
- then, starting from an empty set, insert greedily minimum weight edges in the solution, provided they do not create cycles with the current solution, which, using Tarjan's method, takes another $O((m+n)\alpha(m,n))$ operations, where α is the functional inverse of Ackermann's function.

Now, consider the same problem in a fully dynamic setting, where the set of nodes is fixed, but where edges can be inserted, deleted, or modified at each step. To solve this problem in a more efficient way than recomputing an optimum *ex nihilo* after each graph modification, it suffices to maintain the sorted list from one stage to the other. Unlike building up the sorted list from scratch (which takes $O(m\log(m))$ operations), maintaining it at each stage takes only $O(\log(m))$ operations for each edge insertion, deletion or modification: when an edge is deleted from the list, the list remains sorted; and when an edge is inserted or (resp., modified), one only needs to insert it in (resp., move it to) its right place in the pre-existing sorted list.

Whenever one wants a solution, it only takes $O((m+n)\alpha(m,n))$ operations to compute it out of the sorted list (which is strictly better than $O(m\log(m))$, the "static" complexity of Kruskal's algorithm).

Using this very simple example, one understands how maintaining an underlying data structure through very efficient *update* operations (in this case, maintaining the sorted list) enables to compute solutions at any stage with a *query* operation, with a complexity better than a static algorithm.

3.2. Amortized complexity analysis

We now formally define the problems considered. We are considering a fully dynamic graph G over a fixed vertex set V, $|V| = n$; m generally denotes the current number of edges, which is often assumed to be 0 or $n(n-1)/2$ (so an independent set, or a clique) at the beginning of the algorithm.

Most of the time bounds presented are *amortized*, meaning that they are averaged over all operations performed. This complexity measure is particularly adapted to analyzing on-line or fully-dynamic algorithms, since it enables to catch some features that the worst-case analysis is unable to apprehend. Namely, in a dynamic setting, the algorithm might be designed to avoid encountering the worst case frequently, so that an average measure on all operations allows to "dilute" the occasional worst case complexity in all other -easily updated- cases. Indeed, the analysis amounts to "saving" and "spending" time, just like money on a bank account. The amortized complexity is given by the minimal number of operations one needs at each stage, operations which will be either used or "saved". "Saving" time when the simple case occurs enables to reduce the number of operations needed to handle hard cases, since some operations of the hard cases will be somehow "paid" with all the operations saved previously.

To give a clearer idea of how these so-called amortized complexity analyses are built, we will present a classical example: dynamically resizeable arrays. The process consists in, starting from an array of size 1, to add n elements in this array, by doubling its size whenever it becomes full. The worst case complexity occurs when the array size is doubled because it takes $O(n)$ operations to do this, and considering that n elements are added, the worst case complexity for the overall process is $O(n^2)$.

Here consider that one "saves" two operations when the simple case occurs, namely adding an element when the array is not full. Basically, this case takes only one operation, plus two operations that are saved for later, so in all, 3 operations. Also consider that one "spends" all saved operations when the complex case occurs, namely adding an element when the array is full. Basically this case takes $m+1$ operations considering that m is the size of the full array that has to be doubled, plus the two saved operations, so in all, $m+3$ operations. Whenever the complex case occurs, m operations have been saved ($m/2$ new elements have been added since the last resize), if one "spends" them all, one only needs 3 additional operations to solve the hard case.

Thus, the amortized complexity is 3, so $O(1)$, far better than the worst case complexity $O(n)$. Finally, the amortized analysis enables us to assert that the overall complexity for inserting n elements in a dynamic array is $O(n)$, not $O(n^2)$.

3.3. A restriction of the fully dynamic setting: on-line optimization

It would be hard not to mention on-line optimization while surveying different models of dynamic environments in combinatorial optimization. However, considering the huge flow of literature tackling optimization problems in this framework, we will only say a few words about it, and refer the interested reader to full surveys, such as [2], or [117], focusing on scheduling problems.

On-line models and algorithms have much in common with fully dynamic ones. Indeed, in the online framework, it is supposed that an instance I of a problem Π is revealed gradually, and an online algorithm `OA` is asked to maintain a feasible solution at all times. A natural requirement for `OA` is that it eventually produces a solution as close to $\text{opt}(I)$ as possible. To this point, an online problem is the same as a fully dynamic one, restricted to cases where no data is allowed to be deleted.

But the online model is actually much more restrictive. The main difficulty regarding online problems, is that at each step, one must decide if the new data will be added in the solution or not. If it is not, then it is irrevocably lost, and cannot be taken back at a later stage of the online process. Thus, it is generally impossible to guarantee optimality, and most studies focus on competitive analysis.

Competitive analysis is a method designed especially for analyzing online algorithms, in which the performance of an online algorithm is compared to the performance of an optimal off-line algorithm. An algorithm is said to be competitive if its competitive ratio – the ratio between the value of the solution it eventually returns and the value of an optimum returned by an off-line algorithm – is bounded.

In competitive analysis, one imagines that the instance is built and revealed by an "adversary" that deliberately chooses difficult data, to maximize the competitive ratio. Classically, the adversary will try to prevent the online algorithm from including elements of the optimum, by designing a specific instance and data-sequence.

On-line models help representing and optimizing numerous practical problems, and quite naturally, many NP-hard problems were tackled in this setting. A particular class of problems which online versions were intensively discussed are BIN PACKING [109, 123], scheduling problems [**?**], but other problems such as MIN COLORING [**?**], MAX INDEPENDENT SET [49, 50, 68], TSP [85**?**], or MIN VERTEX COVER [47] were also discussed in online settings, the list is not exhaustive.

3.4. Polynomial problems in dynamic graphs

Since the first paper by Frederickson [58] presenting a data structure known as *topology trees* handling the fully dynamic versions of CONNECTIVITY and of MAXIMUM SPANNING FOREST problem in $O(m)$ operations per update, many improvements have been made on this result [54,72,75], and many other polynomial problems were tackled in the fully dynamic setting, such as SHORTEST PATH [52,62,92], MIN SPANNING TREE [75,119], 2-EDGE CONNECTIVITY [54,59,73,75], and BICONNECTIVITY [70,71,74,75]. Meanwhile, some lower-bounds complexities for these various problems were also provided [98]. Instead of detailing all these results, we will focus on two major problems, CONNECTIVITY and SHORTEST PATH.

3.4.1. CONNECTIVITY problem

Given a graph $G(V,E)$ the CONNECTIVITY problem consists in deciding whether the graph is connected or not. Considering the implications it has on all related problems (while updating data structures of various fully dynamic problems, one often has to know whether the structure is connected), it is not surprising that the CONNECTIVITY problem received a specific attention in the fully dynamic setting. Many successive improvements were provided since the first results by Frederickson in the early 80's. We will sketch out the technique used by Holm *et al.* [75], providing the best complexity known for it, which also enabled the authors to improve computational costs for three other connectivity-related problems.

Theorem 3.1 ([75]). *Given a graph G with m edges and n vertices, there exists a deterministic fully dynamic algorithm that answers connectivity queries in $O(\log n/\log\log n)$ worst-case time, and uses $O(\log^2 n)$ amortized time per insert or delete.*

Leaving aside the worst-case time for connectivity queries, which requires a very technical proof, we will focus on the $O(\log^2 n)$ amortized time, improving the previous $\sqrt[3]{n}\log n$ by Henzinger and King [72]. The idea is the following: using Frederickson's technique [58], one only has to (re)compute a spanning forest at each step, which will drastically decrease the computational cost of connectivity queries. Keeping a spanning forest updated on a dynamic graph can be complicated only when an edge removal disconnects a tree of the spanning forest:

- when an edge (v,w) is inserted, either v and w are connected in the current spanning forest F (one can check this with low computational cost) and (v,w) is not added to the spanning forest, or v and w are not connected in F, and one must add (v,w) to F, to keep it maximal;
- when an edge (v,w) is deleted, it might disconnect the graph only if $(v,w)\in F$. In this case, one has to replace it with another edge, if any. In terms of worst-case complexity, this might be very costly, since one has to check every possible edge to be sure that there is no such reconnecting edge.

But recall that the measure we are interested in is *amortized* time. Holm *et al.* provide a method to decrease it down to $O(\log^2 n)$. Basically, the technique used to decrease the complexity is to assign a *level* $l(e) \leq l_{\max} = \lfloor \log_2 n \rfloor$ to each edge e of the graph. For each i, F_i denotes the sub-forest of F induced by edges of level at least i. Thus, $F \subseteq F_0 \subseteq F_1 \subseteq \ldots \subseteq F_{l_{\max}}$.

At the beginning of the algorithm, every edge is assigned the level 0, and so is every new inserted edge. Edge levels increase by one whenever they belong to a tree that is disconnected due to an edge deletion, or each time they are being candidate as reconnecting edges. Without getting too much into detail regarding how the algorithm maintains these properties true at all time, the main properties that bounds the amortized cost per update are the following:

- F is a maximal (with respect to l) spanning forest of G, that is, if (v, w) is a non-tree edge, v and w are connected in $F_{l(v,w)}$;
- the maximal number of vertices in a tree in F_i is $\lfloor n/2^i \rfloor$.

The first property bounds the number of edges to be considered as candidates for reconnecting whenever an edge $e \in F$ is deleted, while the second ensures that $l_{\max} = \lfloor \log_2 n \rfloor$ as stated earlier.

The authors derive also results for MAXIMUM SPANNING FOREST (result derived directly), MIN SPANNING TREE (derived from an application of the method, but in a graph that is connected from the beginning, instead of an empty graph, and taking weights in account), 2-EDGE CONNECTIVITY and BICONNECTIVITY.

3.4.2. ALL PAIRS SHORTEST PATH

In a connected graph $G(V, E)$ the ALL PAIRS SHORTEST PATH consists in finding shortest paths between all pair of nodes of G, thus $n(n-1)/2$ shortest paths. This problem was one of the very first to be analyzed in the dynamic setting [96, 105, 112]. Although the literature dealing with this topic has been rather rich, it is only in the early 90's that the first provably efficient dynamic algorithm was designed. Ausiello *et al.* [6] proposed a decrease-only shortest path algorithm for directed graphs having positive integer weights less than C: the amortized running time of their algorithm is $O(Cn \log n)$ per edge insertion (or weight decrease, which generalizes insertion).

As an example, we sketch the ideas of the best fully dynamic algorithm for the problem to our knowledge [52]. The authors enhance the previous best known results for both increase-only and fully dynamic cases.

Theorem 3.2 ([52]). *In an increase-only sequence of $\Omega(m/n)$ operations, if shortest paths are unique and edge weights are non-negative, there exists a fully dynamic algorithm that answers each update operation in $O(n^2 \log n)$ amortized time, and each distance and path query in optimal time. The space used is $O(mn)$.*

The main idea enabling this complexity is the use of a specific notion, called "locally shortest path", which applies to all paths Π_{xy}, for which all proper subpaths are shortest paths. Let SP and LSP be the sets of shortest paths, and locally shortest paths respectively. One can directly affirm that $SP \subseteq LSP$, and it is easy to verify if a path $\Pi_{xy} \in LSP$ or not. A given path is locally shortest if and only if:

- it is a single edge; or
- both paths from the first vertex to the last but one vertex, and from the second to the last are shortest paths.

The authors prove that the update of both sets SP and LSP can be done in $O(n^2 \log n)$ amortized time, proving that no more than n^2 paths can stop being locally shortest paths after an update, and using the existing knowledge of the set LSP before an update to compute the modified set SP after the update in an efficient way. Moreover, it is proved that it cannot be more than mn locally shortest paths in a given graph, from which one derives the space bound immediately.

The method and underlying data structures are then adapted to the fully dynamic setting (handling both increases and decreases).

Theorem 3.3 ([52])**.** *In a fully dynamic sequence of $\Omega(m/n)$ operations, if shortest paths are unique and edge weights are non-negative, there exists a fully dynamic algorithm that answers each update operation in $O(n^2 \log^3 n)$ amortized time, and each distance and path query in optimal time. The space used is $O(mn \log n)$.*

The technique used is basically the same up to the fact that the notion of "locally shortest path" is extended to the notion of "locally historical path", *i.e.*, a path whose proper sub-paths are historical, meaning that they have been shortest paths at least once during the process. Of course, the number of locally historical paths being larger than the number of locally shortest paths, the amortized time per update goes up to $O(n^2 \log^3 n)$.

These two examples show how proper data structures can improve amortized time complexity when dealing with polynomial problems in a fully dynamic setting. A whole different picture appears when applying this setting to NP-hard problems.

3.5. NP-hard problems in dynamic graphs

Recall that, in the fully dynamic setting, the process always starts with trivial instances (for example, graph problems would start with independent sets, or cliques), which are modified locally at each step, and the goal is to maintain a given data structure at each step. This data structure helps computing good solutions in a more efficient way than when dealing with a "raw" instance.

In Section 3.4, we presented the kind of data structures and results that were proposed in the literature when applying the fully dynamic setting to polynomial problems. In that case, the data structures helped computing optimal solution in a more efficient way than computing new optima from scratch after each local modification. On the opposite, when applying this setting to NP-hard problems, the data structures will help maintaining approximate solutions.

Moreover, when applying this setting to NP-hard problems, one cannot expect better approximation ratios than in the static case. Indeed, any instance of a given problem Π can be built out of a trivial instance modified locally a polynomial number of times. Thus, a given data structure maintaining an approximation ratio ρ after each local modification can also be seen as a static algorithm providing ρ-approximate solutions: starting from a trivial instance, modifying this instance a polynomial number of times, and updating the data structure after each modification, one can get a ρ-approximate solution on any instance in polynomial time.

Thus, in what follows, we will present dynamic algorithms which aim maintaining approximate solutions, more efficiently than static approximation algorithms, but not providing better ratios.

Only three NP-hard problems were proposed approximation algorithms in this setting: BIN PACKING [79], MIN VERTEX COVER [78], and SHORTEST PATH IN PLANAR GRAPHS [93] which are all rather well approximated problems. In the following, we will provide some details for the two first problems.

3.5.1. BIN PACKING

One of the most elementary and easy to approximate problem, the BIN PACKING problem consists, given a list of items $L = \{a_1, a_2, \ldots, a_n\}$ of size $s(a_i) \in (0, 1]$, in finding the minimum k so that all the items fit into k unit-size bins.

This problem is known to be hard to approximate within $3/2 - \epsilon$. If such an approximation exists, one could partition n non-negative numbers into two sets of minimal size in polynomial time, and this problem is also known to be NP-hard. However, one understands that this inapproximability occurs only for instances with rather small optima, so that approximation algorithm providing better and better *asymptotical* approximation ratios were proposed for this problem. The problem was shown to be asymptotically in PTAS [89], and the best practical algorithm provides an asymptotical approximation ratio of $71/60$ in $O(n \log n)$ [88], generating a solution that uses no more than $(71/60)\mathrm{opt} + 1$ bins, where opt is the optimal number of bins.

The main problem when dealing with BIN PACKING in a fully dynamic setting, is that classical off-line methods do not resist insertion of new items, since these methods rely on sorting all the items in decreasing size order, and then using the sorted list to build the packing. For instance, placing items in the first bin where they fit provides a $11/9$ approximation ratio using a sorted item list. It is obvious that this kind of method is not adapted to the dynamic setting.

Using an adaptation of Johnson's approximation algorithm [86,87], Ivković and Lloyd [79] provide a fully dynamic algorithm achieving a competitive ratio almost as good as the best static approximation algorithm.

Theorem 3.4 ([79]). *The fully dynamic bin packing algorithm, called* `MMP`*, (Mostly Myopic Packing) is asymptotically* $5/4$*-competitive and requires* $O(\log n)$ *per insert/delete operation.*

The idea of Johnsons's algorithm is to regroup items in five size groups: *Big*, *Large*, *Small*, *Tiny* and *Minuscule*. *Big* contains items having size in $]1/2, 1]$ and

Minuscule items having size in $]0, 1/5]$ (other thresholds being logically $1/3$ and $1/4$).

Intuitively, this structure is much more adapted to the dynamic setting than a sorted list on all items. Ivković and Lloyd's algorithm achieves its competitive ratio by using the technique whereby the packing of an item is done with a total disregard for already packed items of a smaller size (*i.e.*, belonging to smaller-size groups).

Each of these myopic packings may then cause several smaller items to be repacked (in a similar fashion). Indeed, an item might be packed in a bin where it does not fit, due to the presence of smaller size items in the bin, which need to be repacked in other bins. With some additional sophistication to avoid certain "bad" cases where most of the smaller size items need to be repacked in cascading sequence, the number of items (either individual items or "bundles" of very small items treated as one item) that need to be repacked is bounded by a constant, thus bounding both the competitive ratio and complexity.

3.5.2. MIN VERTEX COVER

A vertex cover of a graph $G(V, E)$ is a set of vertices such that each edge of the graph is incident to at least one of them. The problem of finding a minimum vertex cover is paradigmatic in computer science and is approximable within ratio 2. Indeed, Gavril [13] proved that taking all endpoints of a maximal matching results in a 2-approximate vertex cover.

This method happens to be rather "dynamic-friendly", but can hardly handle some pathological case in sparse graphs: while insertion of edges, and deletion of edges not in the current maximal matching M can be both handled $O(1)$, deletion of edges of M might require to scan the whole vertex set in order to recompute a maximal matching, thus requiring $O(m)$ operations, which is also the off-line case complexity. However, this dynamic method behaves rather well in dense graph, and will be referred to as `Clean` in what follows.

Ivković and Lloyd's [78] propose to maintain at all time an additional bit for each vertex, indicating if the vertex is matched or unmatched in the current maximal matching. This enables to reduce the worst-case complexity of recomputing a maximal matching from $O(m)$ to $O(1)$, while $O(m)$ operations are needed to maintain the lists of matched and unmatched vertex updated. The key idea of the `Dirty` method is to update these lists only once, thus reducing the amortized complexity and keeping a bounded number of "inaccuracies" in the lists. This algorithm behaves well in sparse graphs.

But actually, in the dynamic setting, the graph might turn from sparse to dense and vice versa, after a certain amount of updates, so that both `Clean` (improving complexity in dense graphs) and `Dirty` (improving complexity in sparse graphs) might end up confronted to their respective pathological cases. The overall algorithm of Ivković and Lloyd, denoted `B1`, tests frequently the density of the graph (which is evolving along with each update), and decides to use either `Clean` or

`Dirty`, depending on whether the current is dense or sparse, in order to ensure an overall enhanced complexity.

Theorem 3.5 ([78]). *`B1` is a 2-competitive fully dynamic approximation algorithm for vertex cover (thus, it is approximation-competitive with the standard off-line algorithm). Further, both insert and delete operations require $O((n+m)^{3/4}$ amortized running time.*

It is clear that maintaining a bounded competitive ratio in fully dynamic setting is quite a hard task when dealing with NP-hard problems. This difficulty arises from needing to compute an approximate solution based on another approximate solution, and this seems to be only possible in very specific cases, and with well approximable NP-hard problems. To cope with this "resistance" to dynamic approximation of some NP-hard problems, and also to answer to numerous practical applications, a restriction of the fully-dynamic setting was introduced, the *reoptimization*.

4. Reoptimization: general properties

As mentioned earlier, if one wishes to get an approximate solution on the perturbed instance, computing it by running a known approximation algorithm from scratch on the modified instance is always possible. In other words, reoptimizing is at least as easy as approximating, since at worst, the information on the initial optimum can remain unused, which amounts to the static problem. Thus, the goal of reoptimization is to determine if it is possible to fruitfully use our knowledge on the initial instance in order to:

- either achieve better approximation ratios;
- or devise faster algorithms;
- or both.

In this section, we present some general results dealing with reoptimization properties of some NP-hard problems, many of which have already been presented in [10]. We first give some general approximation results on the class of hereditary problems presented in [10, 40, 41]. Then, we discuss the differences between weighted and unweighted versions of classical problems, and finally present some ways to achieve hardness results in reoptimization, using the example of MIN COLORING.

Before presenting properties and results regarding reoptimization problems, we will first give formal definitions of reoptimization problems, instances, and approximate reoptimization algorithms. We first give a basic definition of what is an NP optimization problem (NPO).

Definition 4.1. A problem Π in NPO is a quadruple $(\mathcal{I}_\Pi, Sol_\Pi, m_\Pi, goal(\Pi))$ where:

- $\mathcal{I}_\Pi$ is the set of instances of Π (and can be recognized in polynomial time);

- given $I \in \mathcal{I}_\Pi$, $Sol_\Pi(I)$ is the set of feasible solutions of I, the size of a feasible solution of I is polynomial in the size $|I|$ of the instance; moreover, one can determine in polynomial time if a solution is feasible or not;
- given $I \in \mathcal{I}_\Pi$, and $S \in Sol_\Pi(I)$, $m_\Pi(I, S)$ denotes the value of the solution S of the instance I, m_Π is called the objective function, and is computable in polynomial time;
- $goal(\Pi) \in \{\min, \max\}$.

It is easy to see that via Definition 4.1, the class NPO is the class of optimization problems whose decision counterparts are in NP. We now define a reoptimization problem.

Definition 4.2. A reoptimization problem $R\Pi$ is given by a pair $(\Pi, RR\Pi)$ where:

- Π is an optimization problem as defined in Definition 4.1;
- $R_{R\Pi}$ is a rule of modification on instances of Π, such as addition, deletion or alteration of a given amount of data. Given $I \in \mathcal{I}_\Pi$ and $R_{R\Pi}$, $modif_{R\Pi}(I, R_{R\Pi})$ denotes the set of instances resulting from applying modification $R_{R\Pi}$ to I. Notice that $modif_{R\Pi}(I, R_{R\Pi}) \subset \mathcal{I}_\Pi$.

For a given reoptimization problem $R\Pi(\Pi, R_{R\Pi})$, a reoptimization instance $I_{R\Pi}$ of $R\Pi$ is given by a triplet (I, S, I'), where:

- I denotes an instance of Π, referred to as the "initial" instance;
- S denotes a feasible solution for Π on the initial instance I;
- I' denotes an instance of Π in $modif_{R\Pi}(I, R_{R\Pi})$. I' is referred to as the "perturbed" instance.

For a given instance $I_{R\Pi}(I, S, I')$ of $R\Pi$, the set of feasible solutions is $Sol_\Pi(I')$.

Definition 4.3. For a given optimization problem $R\Pi(\Pi, R_{R\Pi})$, a reoptimization algorithm `A` is said to be a ρ-approximation reoptimization algorithm for $R\Pi$ if and only if:

- `A` returns a feasible solution on all instances $I_{R\Pi}(I, S, I')$;
- `A` returns a ρ-approximate solution on all reoptimization instances $I_{R\Pi}(I, S, I')$ where S is an optimal solution for I.

Note that Definition 4.3 is the most classical definition found in the literature, as well as the one used in this paper. However, an alternate (and more restrictive) definition exists (used for example in [28,35,36]), where a ρ_1-approximation reoptimization algorithm for $R\Pi$ is supposed to ensure a $\rho_1\rho_2$-approximation on any reoptimization instance $I_{R\Pi}(I, S, I')$ where S is a ρ_2 approximate solution in the initial instance I.

4.1. Reoptimizing hereditary problems under vertex insertion ...

A property $\mathcal{P}$ on a graph is hereditary if the following holds: if the graph satisfies $\mathcal{P}$, then $\mathcal{P}$ is also satisfied by all its induced sub-graphs. Following this definition, stability, planarity, bipartiteness are three examples of hereditary properties. On the other hand, connectivity is no hereditary property since there might exist some subsets of G whose removal disconnect the graph. Note that an alternative definition of hereditary property in graphs can be provided by the following theorem.

Theorem 4.4 ([110]). *Any hereditary property in graphs can be characterized by a set of forbidden induced sub-graphs.*

In other words a property $\mathcal{P}$ is hereditary if and only if, there is a set H such that every graph that verifies $\mathcal{P}$ is H-free. For instance:

– an independent set is characterized by one forbidden sub-graph: 2 connected vertices;
– a planar graph is characterized by an infinite set of forbidden sub-graphs: all sub-graphs that admit a K_5 (a clique on 5 vertices), or a $K_{3,3}$ as *minor* [125];
– a bipartite graph is also characterized by an infinite set of forbidden sub-graphs: all odd cycles.

Now, let us define problems based on hereditary properties. Let us note that:

Definition 4.5. Let $G = (V, E, w)$ be a vertex-weighted graph. We call Hered the class of problems consisting, given a graph $G = (V, E)$, in finding a subset of vertices S such that $G[S]$ satisfies a given hereditary property and that maximizes $w(S) = \sum_{v \in S} w(v)$.

For instance, MAX WEIGHTED INDEPENDENT SET, MAX WEIGHTED INDUCED BIPARTITE SUB-GRAPH, MAX WEIGHTED INDUCED PLANAR SUB-GRAPH are three classical problems in Hered that correspond to the three hereditary properties given above.

In what follows, G will denote the initial graph, opt a known optimal solution on G, G' the modified instance (resulting from a local modification on G), and opt$'$ an optimal solution on G'. Under vertex-insertion (where G is a sub-graph of G'), there are three powerful properties that one can use when reoptimizing problems in Hered:

(i) the initial optimum opt remains a feasible solution in the modified graph G'; this derives directly from the definition of an hereditary property;
(ii) the part of the new optimum opt$'$ induced by vertices of the initial graph cannot exceed the initial optimum; otherwise, this part would be a better solution than the initial optimum in the initial graph;
(iii) a single node always verifies a hereditary property; this derives also directly from the characterization of hereditary properties in terms of forbidden minors.

Considering these three properties, one can formulate a general algorithm to approximate any weighted problem in Hered within ratio 1/2. Let `R1` denote the generic algorithm which consists in returning the best solution between the initial optimum opt (denoted by S_1) and the single inserted node x (denoted by S_2).

Proposition 4.6 ([10]). *Under vertex insertion,* `R1` *approximates any problem Π in Hered within ratio 1/2 in constant time.*

Proof. The result is quite straightforward when considering the three properties stated above: while properties (i) and (iii) assert that `R1` returns a feasible solution, property (ii) can be reformulated with the following bound:

$$w(S_1) \geq w(\text{opt}') - w(S_2) \tag{4.1}$$

from which one derives directly the approximation ratio. □

Recall that S_1 consists of a single node, so that one should be able to complete it with some other vertices of the graph. In particular, one could run an approximation algorithm on the "remaining instance after taking x". Consider for instance MAX WEIGHTED INDEPENDENT SET, and revisit the proof of the previous proposition. If opt′ does not contain x, then the initial solution opt is optimal. If, in the opposite, opt′ contains x, then consider the remaining graph after having removed x and its neighbors. Suppose that one uses an approximation algorithm which guarantees a ρ-approximation ratio on the remaining graph, and that one adds x to this approximate solution. Denote this generic algorithm `R2`. Then the so-obtained solution S'_2 verifies:

$$w(S'_2) \geq \rho(w(\text{opt}') - w(x)) + w(x) = \rho w(\text{opt}') + (1-\rho)w(x). \tag{4.2}$$

On the other hand, it still holds:

$$w(S_1) \geq w(\text{opt}') - w(x). \tag{4.3}$$

Denoting by S the best solution among S_1 and S'_2, adding (4.2) and (4.3) with coefficients 1 and $(1-\rho)$, one gets:

$$w(S) \geq \frac{1}{2-\rho} w(\text{opt}'). \tag{4.4}$$

which is better than ρ.

Even if not directly applicable to problems in Hered (most of them have been proved to be inapproximable unless P=NP [97]), the technique is rather general, and can find applications in the reoptimization of many problems. We illustrate it in two well-known problems: MAX WEIGHTED SAT (Thm. 4.7) and MIN VERTEX COVER (Thm. 4.8).

Given a conjunction of weighted clauses over a set of binary variables, MAX WEIGHTED SAT asks for the truth assignment of variables maximizing the sum of weights of satisfied clauses.

Theorem 4.7 (([10]). *Under the insertion of a clause, reoptimizing* MAX WEIGHTED SAT *is approximable within ratio 0.81.*

Proof. Consider a conjunction of clauses φ over a set of binary variables, each clause being given with a weight, and let τ^* be an initial optimum solution. Consider that a new clause $c = (l_1 \vee l_2 \vee \ldots \vee l_k)$, (where l_i is a literal of variable x_i, *i.e.*, either x_i or $\bar{x}_i$) has weight $w(c)$. The modified formula is thus given by $\varphi_c = \varphi \cup \{c\}$. Then, k different solutions τ_i, $i = 1, \ldots k$ are computed. Each τ_i is built as follows:

– set l_i to true;
– delete from φ all satisfied clauses;
– apply a ρ-approximation algorithm to the remaining instance (note that the clause c is already satisfied); together with l_i, this is a particular solution τ_i.

Finally, the best solution among all τ_i's and the initial optimum τ^* is returned.

As previously, if the optimal solution τ_c^* on the modified instance does not satisfy c, then τ^* remains optimal for the new problem. Otherwise, at least one literal in c, say l_i, is true in τ_c^*. Considering that l_i is true in τ_i, it is easy to see that:

$$w(\tau_i) \geq \rho(w(\tau_c^*) - w(c)) + w(c) = \rho w(\tau_c^*) + (1-\rho)w(c). \tag{4.5}$$

Once more, as in the general technique described above:

$$w(\tau^*) \geq w(\tau_c^*) - w(c). \tag{4.6}$$

So, (4.4) holds for MAX WEIGHTED SAT also. Taking into account that this problem is approximable within ratio $\rho = 0.77$ [5], the result claimed is concluded. □

Let us now focus on a minimization problem, namely MIN VERTEX COVER. Given a vertex-weighted graph $G = (V, E, w)$, the goal is to find a subset $S \subseteq V$ such that every edge $e \in E$ is incident to at least one vertex in S, and that minimizes $w(S) = \sum_{v \in S} w(v)$.

Theorem 4.8 ([10]). *Under a vertex insertion, reoptimization of* MIN VERTEX COVER *is approximable within ratio* 3/2.

Proof. Let x denote the new vertex and opt the initial given solution. Then, $\text{opt} \cup \{x\}$ is a vertex cover on the final instance. If opt' takes x, then $\text{opt} \cup \{x\}$ is optimum.

Suppose now that opt' does not take x. Then, it has to take all its neighbors $N(x)$. Observe that $\text{opt} \cup N(x)$ is a feasible solution on the final instance, since one only has to add all the vertices needed to cover edges incident to x, every other edge being already covered in the initial optimum. Since $w(\text{opt}) \leq w(\text{opt}')$, we get:

$$w(\text{opt} \cup N(x)) \leq w(\text{opt}') + w(N(x)). \tag{4.7}$$

Then, as for MAX WEIGHTED INDEPENDENT SET, consider the following feasible solution S1:

- take all the neighbors $N(x)$ of x in S_1;
- remove x and its neighbors from the graph;
- apply a ρ -approximation algorithm on the remaining graph, and add the resulting vertex cover to S_1.

Recall that opt′ does not take x, and thus has to take all its neighbors, we finally get:

$$w(S_1) \leq \rho(w(\text{opt}') - w(N(x))) + w(N(x)) = \rho(w(\text{opt}')) + (1-\rho)(N(x)). \quad (4.8)$$

Outputting S the best solution between, $\text{opt} \cup N(x)$ and S_1 a convex combination of (4.7) and (4.8) leads to:

$$w(S) \leq \frac{2\rho - 1}{\rho} w(\text{opt}').$$

The result follows since MIN VERTEX COVER is well-known to be approximable within ratio 2. □

For MAX WEIGHTED INDEPENDENT SET, the 1/2 approximation ratio provided by algorithm `R1` is actually the best approximation ratio achievable in polynomial time. Indeed, the following proposition is proved in [40].

Proposition 4.9 ([40])**.** *Under vertex insertion, reoptimizing* MAX WEIGHTED SAT *is inapproximable within ratio* $\frac{1}{2} + \varepsilon$ *in polynomial time, unless* P=NP.

The proof is based upon a construction where a static (yet inapproximable within any constant ratio) instance H of MAX WEIGHTED INDEPENDENT SET is a sub-graph of a reoptimization instance $I(H)$. The instance $I(H)$ is built such that a $1/2 + \varepsilon$-approximation algorithm used on it produces a solution S which restriction to H is a ε approximation for MAX WEIGHTED INDEPENDENT SET in H, which is impossible to achieve in polynomial time unless P=NP.

Note also that results of Propositions 4.6 and 4.9 can be generalized to the MAX k-PARTITE SUBGRAPH problem, under the insertion of h vertices.

Proposition 4.10 ([40])**.** *Under insertion of h vertices,* MAX k-PARTITE SUBGRAPH *is approximable within ratio* $\max\left\{\frac{k}{k+h}, \frac{1}{2}\right\}$ *in polynomial time, but it is inapproximable within ratio* $\max\left\{\frac{k}{k+h}, \frac{1}{2}\right\} + \varepsilon$ *in polynomial time, unless* P=NP.

Note that in [40], similar results (approximation ratios and inapproximability bounds) are provided for MAX SPLIT SUBGRAPH, MAX P_k-FREE SUBGRAPH and MAX PLANAR SUBGRAPH.

Note finally that results similar to those of Propositions 4.9 and 4.10 (that is, optimal approximation results) are provided in [29] regarding the reoptimization versions of MAX WEIGHTED INDEPENDENT SET, DOMINATING SET, and SET COVER, under insertion and deletion of edges.

4.2. ...and vertex deletion

Let us consider now the opposite kind of perturbations: vertex-deletion. When dealing with hereditary optimization problems, some properties discussed just above still remain valid, while some others do not so. As before, let us consider a given instance of a problem in Hered, for which we know an optimal solution opt. Consider now that one vertex of the graph is deleted, along with its incident edges. Two cases might occur:

– the deleted vertex x was no part of the initial optimum, so it remains the same in the new graph;
– x was part of the initial optimum, and might even have been one of its most important elements. Though having *a priori* no information on the quality of the initial optimum opt$\setminus\{x\}$ in the new graph G_x (or rather what is left of it), we can still assert that opt$\setminus\{x\}$ remains a feasible solution in the new graph.

To prove that opt$\setminus\{x\}$ remains feasible just observe that, by heredity, opt$\setminus\{x\}$ is feasible in the initial graph and since this graph is an induced sub-graph of the new graph it is also feasible for this latter graph.

In [40], a strong and tight inapproximability bound is proved regarding MAX k-PARTITE SUBGRAPH in general graphs.

Proposition 4.11 ([40]). *Under deletion of $h < k$ vertices,* MAX k-PARTITE SUBGRAPH *is approximable within ratio $\frac{k-h}{k}$ in polynomial time, and inapproximable within ratio $\frac{k-h}{k} + \varepsilon$ in polynomial time, unless* P=NP. *Under deletion of $h \geq k$ vertices,* MAX k-PARTITE SUBGRAPH *is inapproximable within ratio n^{ε} in polynomial time, unless* P=NP.

One directly derives from this result that MAX WEIGHTED INDEPENDENT SET is inapproximable within any constant ratio under vertex deletion (since this problem amounts to MAX 1-PARTITE SUB-GRAPH). Thus, in general graphs, one cannot expect to implement `R1` or `R2` algorithms under vertex deletion.

However, this strong inapproximability result does not extend to graphs of bounded degree. For this class of graphs, the following is proved.

Proposition 4.12. *Under-vertex deletion,* MAX WEIGHTED INDEPENDENT SET *is approximable within ratio* $1/2$ *in graphs of bounded degree.*

The idea of the algorithm achieving this result is the following: it returns the best solution between the remaining part of the optimum, and an optimal solution in the neighborhood of deleted vertices, which can be computed through exhaustive search in $O(2^{h\Delta})$, where h denotes the number of deleted vertices, and Δ the maximum degree in the graph.

This algorithm, which is very similar to `R1`, achieves the same approximation ratio. Note that, for $\Delta \geq 5$, the result beats the classical static approximation ratio of $3/(\Delta + 2)$ [67] for MAX WEIGHTED INDEPENDENT SET in graphs of bounded degree.

It is finally shown in [40] that in graphs of degree bounded by $\varDelta$, the reoptimization version of any hereditary problem that is characterized by a forbidden sub-graph of depth at most d, under deletion of h vertices, is equivalent to the reoptimization version of the same problem under insertion of $(d-1)h\varDelta$ vertices. This fact leads to the following proposition.

Proposition 4.13. *In graphs of degree bounded by $\varDelta$, under deletion of h vertices,* MAX WEIGHTED INDEPENDENT SET *is approximable within ratio $\frac{\varDelta+2}{2\varDelta+1}$ in polynomial time.*

Where the result follows from an implementation of algorithm R2.

4.3. Unweighted problems

In the previous paragraph, we have considered the general cases where vertices have a weight. The weight of the inserted or deleted vertices played a major role in most of our proofs, and intuitively, the worst case occurred when the inserted or deleted elements had weights comparable to the weight of the whole optimal solutions, thus much bigger than all the other weights.

All the problems we focus on are already NP-hard in the unweighted case, *i.e.*, when all data receive weight 1. In this case, the previous approximation results on reoptimization can be easily improved. Indeed, since only one vertex is inserted (resp., deleted), the initial solution (resp., the restriction of the initial graph to the modified graph) has an absolute error of at most one on the final instance:

$$|\mathrm{opt}| \geq |\mathrm{opt}'| + 1.$$

Then, in some sense we do not really need to reoptimize since opt is already a very good solution for the final instance (note also that since the reoptimization problem is NP-hard, we cannot get rid of the constant 1). Dealing with approximation ratio, we derive from this remark, with a standard technique, the following result.

Theorem 4.14 ([10])**.** *Under a vertex insertion (resp., deletion), reoptimizing any unweighted problem in Hered admits a* PTAS.

Proof. Fix a constant ε and set $k = 1/\varepsilon$. The following algorithm computes a solution of size at least $(1-\varepsilon).|\mathrm{opt}'|$ in $O(n^k)$:

- test all the subsets of V_x of size at most k, and let S_1 be the largest one such that $G[S_1]$ satisfies the hereditary property considered;
- return the largest solution S among S_1 and opt.

If opt$'$ has at most $1/\varepsilon$ elements, we found it in step 1. Otherwise, $|\mathrm{opt}'| \geq 1/\varepsilon$ and:

$$\frac{|\mathrm{opt}|}{|\mathrm{opt}'|} \geq \frac{|\mathrm{opt}'|+1}{|\mathrm{opt}'|} \geq 1-\varepsilon$$

that completes the proof. □

To be even more general, this technique applies to any problem for which the initial optimum remains feasible, or at least easily adaptable, and that is *simple*, in the sense of [107], meaning that a solution of value k (k a constant) can be found in polynomial time (in our case in $O(n^k)$).

We provide an example of this generalization to non hereditary problems, with the unweighted version of the problem MAX CUT in graphs of bounded degree.

Theorem 4.15. *Under a vertex insertion (resp., deletion), reoptimizing* MAX CUT *in graphs of bounded degree admits a* PTAS*.*

Proof. First let us state that the initial optimum given by a subset of vertices has an absolute error of at most Δ in the new graph.

$$|\mathrm{opt}| \geq |\mathrm{opt}'| + \Delta.$$

Denote by S_1 the solution consisting in leaving what remains of the optimum as it is, fix a constant ε, set $k = \lceil 1/\varepsilon \rceil$, and run the following algorithm:

– test all the subsets of V_x of size at most $k\Delta$, and let S_2 be the one inducing the biggest cut in G';
– return the largest solution S among S_1 and S_2.

As before, one can assert that if opt′ induces a cut of at most $k\Delta$ edges, then we found it in the first step and the algorithm returns the optimal solution. Thus, if opt′ induces a cut having more than $k\Delta$ edges, then:

$$\frac{|\mathrm{opt}|}{|\mathrm{opt}'|} \geq \frac{|\mathrm{opt}'| + \Delta}{|\mathrm{opt}'|} \geq 1 - \varepsilon$$

and the proof is completed. □

The application of such methods requires that the problem considered is *simple.* A problem is said to be *simple* if it takes $O(n^k)$ to decide if it admits a solution of value k. Of course, there exist some unweighted, non *simple* problems where the method above cannot be applied.

One of the most famous such problems is the MIN COLORING problem. In this problem, given a graph $G = (V, E)$, one wishes to assign a color to each vertex, so that no vertex has a neighbour of the same color. In other words, it consists of partitioning V into a minimum number of independent sets (colors) $S = \{V_1, \ldots, V_k\}$. Under vertex insertion, an absolute error 1 can be easily achieved while reoptimizing. Indeed, consider the initial coloring, and add a new color which contains only the newly inserted vertex. Considering that the new graph cannot be colored with less than $|\mathrm{opt}|$ colors, outputting a solution $S_0 = \mathrm{opt} \cup \{x\}$ does only increase the solution by 1 with respect to the initial optimum, whose cardinality is also a lower bound for the new optimum.

However, deciding whether a graph can be colored with 3 colors is an NP-hard problem. In other words, MIN COLORING is not *simple* which prevents us from using the classical building of a PTAS described earlier. We will discuss the consequence of this fact in the paragraph dealing with hardness and inapproximability in reoptimization.

To conclude this section, we underline the fact that there exist obviously many problems that do not involve weights and for which the initial optimal solution cannot be directly transformed into a solution on the final instance with bounded absolute error. Finding the longest cycle in a graph is such a problem: adding a new vertex along with its incident edges may deeply modify the structure of the graph, and change considerably the size of an optimum solution.

4.4. Hardness of reoptimization and inapproximability results

As mentioned earlier, reoptimizing is about finding out to what extent does the information on the optimum of similar instance makes the problem simpler. However, although often making the problem easier, this information does not allow us a jump in complexity regarding the P *vs.* NP dichotomy; a reoptimization problem is often NP-hard when its underlying static version is so. In most cases, the NP-hardness of reoptimization problems can be established immediately, *i.e.*, without resorting to reductions. For instance consider the reoptimization version $R(\Pi)$ of a NP-hard problem Π for which a trivial instance - say an empty graph or a single node - can be solved optimally in polynomial time (or trivially). Moreover, suppose that $R(\Pi)$ is polynomial, and denote by `A` a polynomial algorithm solving it optimally. Then, starting from the trivial instance and inserting nodes one by one, applying `A` after each insertion could produce an optimum on any graph in polynomial time, contradicting so the NP-hardness of Π.

In other cases, the hardness does not directly derive from this argument, and a usual polynomial time reduction has to be provided. As we will see later, such hardness proofs have been given for instance for some vehicle routing problems.

Let us now focus on the hardness of approximation. From all results stated so far, one understands that reoptimization might often provide better worst-case approximation ratios than for the "static" case. Going one step further, we even presented some reoptimization algorithms breaking inapproximability bounds of their underlying static versions. In this sense, extending inapproximabilty results from static to reoptimization versions of NP-hard problems is not a result which can be established naturally. Actually, few such results are established so far, apart from those presented in Section 4.6.

One method is to exploit (possibly by transforming) the reduction used in the proof of NP-hardness to get an inapproximability bound. Though more difficult to establish than in the usual setting, such proofs have been provided for reoptimization problems, in particular for vehicle routing problems, mainly by introducing very large distances (see Sect. 5.5).

Let us now go back to MIN COLORING. As we have said, it is NP-hard to determine whether a graph is colorable with 3 colors or not. In the usual setting, this leads to an immediate inapproximability bound of $4/3 - \varepsilon$ for any $\varepsilon > 0$. Indeed, an approximation algorithm within ratio $\rho = 4/3 - \varepsilon$ would allow to decide in polynomial time whether graph is 3-colorable or not. Now, we extend this result to the reoptimization version.

Theorem 4.16 ([10]). *Under a vertex insertion, reoptimizing* MIN COLORING *cannot be approximated within a ratio $4/3 - \varepsilon$, for any $\varepsilon > 0$.*

Proof. Suppose that there exists an algorithm `A` reoptimizing MIN COLORING within ratio $4/3 - \varepsilon$ for a given $\varepsilon > 0$ under vertex insertion.

Denote by G_i the sub-graph induced by the set of vertices $V_i = \{v_1, \ldots, v_i\}$, Then starting from an empty graph and inserting nodes one by one, one successively solves the problem on graphs G_1, G_2, etc., up to G_n. While G_i is 2-colorable a 2-coloring can be found in polynomial time. Suppose that, at some point, G_i is 3-colorable but not 2-colorable, and insert v_{i+1}. If G_{i+1} is 3-colorable, then `A` returns a 3-coloring (the proof is identical to that for the inapproximability bound in the static case). Moreover, if G_i is not 3-colorable, then G_{i+1} cannot be 3-colorable either. Hence, starting from the empty graph, and iteratively applying `A`, we get a 3-coloring in polynomial time for any G_i if and only if G_i is 3-colorable, a contradiction. □

This proof is based on the fact that MIN COLORING is not *simple* (following to the definition previously given). A similar argument, leading to inapproximability results in reoptimization, can be applied to other non simple problems, and under other modifications, such as data deletion. It has been in particular applied to a scheduling problem (see Sect. 5). For other optimization problems however, such as MIN TSP in the metric case, finding a lower bound (if any) in approximability seems a challenging task, and is still an open problem to this day.

Let us finally mention another kind of negative results. In the reoptimization setting, we look somehow for a possible stability when slight modifications occur in the instance. We try to measure how much the knowledge of a solution on the initial instance helps to solve the modified one. Hence, it is natural to wonder whether one can find a good solution in the *neighborhood* of the initial optimum solution, or if one has to change almost everything. Do neighbouring instances have neighbouring optimum/good solutions?

As an answer to these questions, several results show that, for several problems, approximation algorithms that only *slightly* modify the initial optimum solution cannot lead to good approximation ratios. Such results are provided in [38] about the MIN SPANNING TREE problem.

5. Reoptimizing NP-hard optimization problems

In the previous section, we have provided some general properties concerning reoptimization, and presented some generic algorithms adaptable to whole classes of problems. We will now focus on three NP-hard problems for which particular reoptimization strategies were proposed, namely MIN STEINER TREE, SHORTEST COMMON SUPERSTRING and KNAPSACK. We also present some inapproximability result on KNAPSACK and on a scheduling problem with forbidden sets.

5.1. MIN STEINER TREE problem

In a weighted graph $G(V, E, w)$ the MIN STEINER TREE problem is a generalization of the MIN SPANNING TREE problem, where the objective is to find a minimum weight sub-tree of a given graph spanning a given subset $R \subseteq V$ of vertices, called terminals. It is assumed that the graph is complete, and the distance function is metric (*i.e.*, $w(x, y) + w(y, z) \geq w(x, z)$ for any three vertices x, y, z): indeed, the general problem reduces to this case by initially computing shortest paths between all pairs of vertices.

This is one of the most famous network design optimization problems. It is NP-hard, and has been studied intensively from an approximation viewpoint. The best known ratio obtained so far is $1 + \ln 3/2 \simeq 1.55$ [111].

Reoptimization versions of this problem have been studied with modifications on the vertex set [28, 36, 55], as well edge weight modifications [28]. In Escoffier *et al.* [55], the modification considered consists in the insertion of a new vertex. The authors study the cases where the new vertex is terminal or non terminal.

Theorem 5.1 ([55])**.** *When a new vertex is inserted (either terminal or not), then reoptimizing the* MIN STEINER TREE *problem can be approximated within ratio* 3/2*.*

The result is then extended to the insertion of several vertices. Let p and q denote the number of inserted non-terminal and terminal vertices, respectively. As long as these two values remain constant, the problem remains approximable within ratio 3/2, though the complexity tends to grow very fast with p. It is interesting to notice that the algorithm achieves an approximation ratio decreasing with q, namely $2 - 1/(q + 2)$.

The basic idea of the algorithms is to merge the initial optimal tree with Steiner trees computed on the set of new vertices. The goal is to find -by exhaustive search- some part of the new optimum. As one might guess, the number of partial Steiner trees computed grows very fast with the number of new non-terminal vertices.

Bäckenhauer *et al.* [36] consider a different kind of instance modification: instead of inserting or deleting a vertex, the authors consider the case where the status of a vertex changes: either a terminal vertex becomes non terminal, or vice versa. The obtained ratio is also 3/2.

Theorem 5.2 ([36])**.** *When the status (terminal/non terminal) of a vertex changes, then reoptimizing the* MIN STEINER TREE *problem can be approximated within ratio* 3/2*.*

Moreover, they exhibit a case where this ratio can be improved. For the sub-problem where the edge weights are restricted to values from $\{1, 2, \ldots, k\}$ for some constant k, there is a PTAS for the reoptimization problem based on changing the status of vertices [36], although the static version of this sub-problem was proved APX-hard in its static version even for $k = 2$ [19].

Bilò *et al.* [28] improved upon these results, and achieved better bounds for the sub-cases when a terminal vertex appears, or disappears: 1.408 for the former and

1.34 for the latter. Dealing with edge weight modifications, they presented algorithms achieving approximation ratios 4/3 and 1.3, respectively for weight increase and weight decrease cases. An interesting survey on all these results (together with results on TSP, presented in the next section) can be found in [35].

Also, some results were provided for one generalization of MIN STEINER TREE consisting in considering sharpened triangle inequality, That is, given a constant β, $1/2 \leq \beta \leq 1$, $w(x,y) + w(y,z) \geq \beta \cdot w(x,z)$ for any triple (x,y,z) of vertices in a complete graph (the graph being metric for $\beta = 1$). In [37], an improved $\frac{1}{2} + \beta$ -approximation algorithm for all types of modification considered is presented. More precisely, the following result is proved.

Theorem 5.3 ([37]). *Let (G,S,c) be a Steiner tree instance where c satisfies the sharpened β-triangle inequality for some $1/2 < \beta < 1$. Then there exists a* PTAS *for the reoptimization variants of the sharpened version of* MIN STEINER TREE*, when the edge-costs are modified and when the status of vertices is changed, i.e., when a terminal becomes a non-terminal and vice versa.*

The $1/2+\beta$ is always achieved by comparing the initial optimum (or an adapted version of it when required) with a minimum spanning tree on all terminal vertices, which is proved to be 2β-approximate solution.

5.2. SHORTEST COMMON SUPERSTRING

Given a substring-free set of strings S (meaning a set of strings where no string of the set is a substring of another one), the SHORTEST COMMON SUPERSTRING problem asks for a shortest common superstring of S, *i.e.*, for a minimum-length string containing all strings from S as substrings. This problem is one of the most prominent hard problems in stringology with many applications, and is known to be NP-hard [64] and even APX-hard [124]. Its best "static" approximation ratio is 2.5 [122].

Reoptimization versions of this classical problem have been studied by Biló *et al.* [30], considering string insertion and deletion, both proved NP-hard. In particular, they devise a simple and efficient approximation algorithm for the insertion case.

Theorem 5.4 ([30]). *Under string insertion,* SHORTEST COMMON SUPERSTRING *is approximable within ratio $11/6 - \varepsilon$, for any $\varepsilon > 0$.*

First, notice that any solution can be characterized by an ordering of the strings in S, in which each maximal substring common to the end of one string and to beginning of its next string is deleted once. Then, the reoptimization strategy consists of testing all $n+1$ possible insertions of the new string in this ordered list, and returns the one inducing the best solution. The idea is rather simple, but the analysis which proves the bound on the approximation ratio is very technical: it consists of a complete and involved case analysis. Note that this 11/6-approximation algorithm outperforms the best 2.5 of the static case [122].

Biló *et al.* [28] also devise a polynomial reoptimization algorithm under string insertion with approximation ratio arbitrarily close to $(2\rho - 1)/\rho$, and provide a similar result under string deletion with approximation ratio $(3\rho-1)/(\rho+1)$, where ρ denotes the approximation ratio guaranteed by a static approximation-algorithm used as subroutine. Since the best known polynomial approximation algorithm for SHORTEST COMMON SUPERSTRING achieves $\rho = 2.5$, they obtain an approximation ratio arbitrarily close to $8/5 = 1.6$ under string insertion and an approximation ratio arbitrarily close to $13/7 < 1.86$ under string deletion.

5.3. KNAPSACK

Suppose we are given a set of n objects $O = \{o_1, \dots, o_n\}$, and a capacity C. Each object o_i in O is associated with a weight w_i and a value v_i. The goal is to choose a subset O' of objects which maximizes its global value without overloading the capacity P.

This problem is known to be (weakly) NP-hard, and admits an FPTAS [77]. Obviously, following the general properties of reoptimization, the reoptimization version admits an FPTAS too. Thus, the authors of [4] are interested in using classical "static" approximation algorithms for KNAPSACK as subroutines for reoptimization algorithms with better approximation ratios but with the same running time. The modification considered consists of the insertion of a new object in the instance.

Though not being a graph problem, it is easy to see that KNAPSACK satisfies the required properties of heredity given earlier. Hence, the strategy `R1` consisting in returning the best solution among the initial optimum and the single newly inserted object returns a 1/2-approximated solution in constant time. Moreover, the strategy `R2` can also be easily implemented, so that if we have a ρ-approximation algorithm, then the implementation of strategy `R2` has ratio $1/(2-\rho)$ [4]. Besides, the authors also show that this bound is tight for several classical approximation algorithms for Max Knapsack.

Finally, studying the issue of sensitivity and neighborhood presented earlier, namely analysing to what extent a local modification of the instance can perturb the optimum in terms of value and/or structure, they show that any reoptimization algorithm that does not consider objects discarded by the initial optimal solution cannot have ratio better than 1/2, this is precisely what `R1` does.

5.4. Scheduling

This is probably one of the most practical problems, and this is precisely due to practical motivations that the problem of recomputing a good solution under instance perturbation has been intensively studied. Most studies in this field have aimed at determining how much a given instance may be changed if it is desired that optimal solutions to the original instance remain optimal. The main results in this so called *post-optimality analysis*, as well as sensitivity analysis may be found in the comprehensive article [65].

In the reoptimization setting, some interesting results were achieved for the problem of scheduling with forbidden sets. In this problem, we are given a set of jobs $V = \{v_1, \ldots, v_n\}$, each having a processing time. The jobs can be scheduled in parallel (the number of machines is unbounded), but there is a set of constraints on these parallel schedules: a constraint is a set $F \subseteq V$ of jobs that cannot be scheduled in parallel (*i.e.*, all of them at the same time). Given a set $\mathcal{F} = \{F_1, \ldots, F_k\}$ of constraints, the goal is to find a schedule that respects each constraint and that minimizes the latest completion time. Many situations can be modeled in this way, such as the m-Machine Problem (for fixed m): it suffices to consider all subsets of V of size at least $m + 1$ as constraints, so that no more than m jobs can be processed in parallel. Hence the problem is NP-complete (and also APX-hard). Schäffter [115] considers reoptimization when either a new constraint F is added to $\mathcal{F}$, or a constraint $F_i \in \mathcal{F}$ disappears. Using reductions from SHORTEST SPLITTING and MIN COLORING problem, he proves the following inapproximability results.

Theorem 5.5 ([115]). *Unless* P=NP*, for any* $\varepsilon > 0$*, reoptimizing the scheduling problem with forbidden sets problem is inapproximable within ratio* $3/2 - \varepsilon$ *under constraint insertion, and* $3/4 - \varepsilon$ *under constraint deletion.*

Under insertion of a new constraint, Schäffter [115] provides a 3/2-approximation algorithm, thus matching the lower bound given in Theorem 5.5. This algorithm consists in a single local modification: it shifts the faster task of the new constraint at the very end of the scheduling to ensure that this new constraint is satisfied (recall that a constraint is not satisfied only when *all* its tasks process at the same time). Notice that the result does not extend to multiple tasks insertions.

5.5. Reoptimization of vehicle routing problems

In the well-known TRAVELING SALESMAN (TSP), the objective is to find a minimum-cost Hamiltonian cycle in a complete graph with edge costs. TSP has been used and studied since the beginning of combinatorial optimization, mostly as a testbed for experimenting a whole array of algorithmic paradigms and techniques. In this sense, it is natural to also consider it from the reoptimization point of view, and we survey here several results concerning its reoptimization under several kinds of perturbations.

Definition 5.6. An instance I_n of the TRAVELING SALESMAN is specified by the distance between every pair of n nodes in the form of an $n \times n$ matrix d, where $d(i, j) \in \mathbb{Z}_+$ for all $1 \leq i, j \leq n$. A feasible solution for I_n is a tour, that is, a directed cycle spanning the node set $N = \{1, 2, \ldots, n\}$.

Note that the above definition only specifies the general structure of instances and feasible solutions for TRAVELING SALESMAN. The reason why we do not specify an objective function is that one might use this general structure along with different kinds of objective functions, thus defining as many different problems.

Before presenting the different problems surveyed in what follows, we give some additional definitions:

- the *weight* of a tour T is the quantity $w(T) = \sum_{(i,j)\in T} d(i,j)$;
- the *latency of a node* i with respect to a tour T is the total distance along T from node 1 to node i;
- the *latency of a tour* T, denoted by $l(T)$, is the sum of the latencies of the nodes in T;
- the matrix d satisfies the triangle inequality if, for all $i, j, k \in N$ it holds that $d(i,j) \leq d(i,k) + d(k,j)$;
- the matrix d is said to be metric if it satisfies the triangle inequality and $d(i,j) = d(j,i)$ for all $i, j \in N$.

In what follows, we will survey approximation algorithms for the following versions of the TRAVELING SALESMAN:

(1) MIN TSP: find a tour of minimum *weight*;
(2) MIN METRIC TSP: restriction of MIN TSP to the case when d is metric;
(3) MIN ATSP: restriction of MIN TSP to the case when d satisfies the triangle inequality;
(4) MAX TSP: find a tour of maximum weight;
(5) MAX METRIC TSP: restriction of MAX TSP to the case when d is metric;
(6) MIN LATENCY: find a tour of minimum *latency*, when d is assumed to be metric.

Other variants of TSP have also been studied in a reoptimization setting. In particular, inapproximability results have been given in [32], where the reoptimization versions of TSP with deadlines under various modifications are shown to be $2-\varepsilon$-inapproximable.

Considering a vehicle routing problem Π of the above list, three different reoptimization settings have been considered in the literature: under node insertion, the problem will be referred as $\Pi+$; under node deletion, it will be referred to as $\Pi-$, and $\Pi\pm$ will denote a reoptimization instance where the perturbation consists in modifying one single distance in the matrix d. Note that none of the modifications considered should perturb the "nature" of the distance matrix, *i.e.*, the new instance must satisfy the triangle inequality (resp., be metric) if the initial instance does (resp., is so).

Some simple solution methods are common to several of the problems we study in this section. We define here two such methods which will be used in what follows.

Algorithm 5.7 (nearest insertion)**.** *Given an instance I_{n+1} and a tour T on the set $\{1,\ldots,n\}$, define the node i^* as $i^* \in \operatorname{argmin}_{1\leq i\leq n} d(i,n+1)$. Compute two solutions by inserting $n+1$ immediately after i^* in T, and immediately before i^*, and return the best between them.*

Algorithm 5.8 (best insertion)**.** *Given an instance I_{n+1} and a tour T on the set $\{1,\ldots,n\}$, define the pair (i^*,j^*) as $(i^*,j^*) \in \operatorname{argmin}_{(i,j)\in T} d(i,n+1) + d(n+1,j) - d(i,j)$. Obtain the solution by inserting node $n+1$ between i^* and j^* in the tour.*

5.5.1. TSP

General case. Due to a classical reduction from the HAMILTONIAN CYCLE problem, it is well known that TSP cannot be approximated within any ratio polynomial in the size of the instance, and *a fortiori* within any constant ratio. This result extends to any reoptimization version of the problem, and is proved somewhat in the same way.

Theorem 5.9 ([8, 9, 33]). *Let be p a polynomial. Then all* MIN TSP+*,* MIN TSP− *and* MIN TSP± *cannot be approximated with $2^{p(n)}$, unless* P=NP.

Proof. We will only provide the proof for MIN TSP+, the corresponding results for the other problems are proved with the same technique, namely the *gap technique* from Sahni and Gonzalez [114]. We will reduce an instance of $\text{SHP}_{a,b,s,t}$ (proved to be NP-complete in [8, 9]) to a $2^{p(n)}$-approximation of MIN TSP+. Given a graph $G = (V, E)$ where $a, b, s, t, \in V$, and an Hamiltonian path of G from a to b, $\text{SHP}_{a,b,s,t}$ problem consists in determining whether there exists a Hamiltonian path from s to t in G.

Starting with an instance I of $\text{SHP}_{a,b,s,t}$, an instance I' of MIN TSP+ is built as follows:

- if $(v_i, v_j) \in E$, then $d(v_i, v_j) = 1$;
- $d(a, b) = 1$, so that a initial optimum is given by the Hamiltonian path in E from a and to b, plus the edge (a, b);
- $d(v_{n+1}, s) = d(v_{n+1}, t) = 1$;
- all the other edges have a weight $2^{p(n)} \cdot (n + 1) + 1$.

Of course, a Hamiltonian path from s to t in G can be found in the instance if and only if a Hamiltonian cycle of weight $(n + 1)$ can be found in I'. Moreover, if no such solutions exists, then the optimal Hamiltonian cycle in I' must use at least one edge not in I each having a distance of $2^{p(n)}(n + 1) + 1$, which is thus a lower bound for its weight in this case.

Now, consider a $2^{p(n)}$-approximation algorithm. Such an algorithm would return a solution of value at most $2^{p(n)}(n + 1)$ if the a Hamiltonian path from s to t in G can be found in I, otherwise it should return the optimum in the best case, so a solution of weight at least $2^{p(n)}(n + 1) + 1$. Thus, it would enable us to decide $\text{SHP}_{a,b,s,t}$ in polynomial time. □

Metric case. This case, that is very natural and meets many practical applications, enables constant-ratio approximation. Christofides [46] proposed in 1976 a (3/2)-approximation algorithm that consists in merging a minimum spanning tree and a minimum matching on its odd-degrees vertices (both computable in polynomial-time). No better approximation algorithm has been found since, but it is still an open problem to determine whether this 3/2 can be outperformed or not. Notice finally that the metric MIN METRIC TSP has been proved to be APX-hard [106].

This restriction to metric graphs allows better results in reoptimization setting, in particular under node insertion and distance modification.

Theorem 5.10 ([8, 9])**.** *In the metric case,* MIN METRIC TSP+ *is approximable within ratio* 4/3*.*

Proof. The result derives from an algorithm which produces a solution T_1 with the *Nearest Insertion* procedure, another one with Chistofides' algorithm, say T_2 and returns the best among them.

One of the main ideas of Christofides' algorithm is to start with a minimum spanning tree on the input graph, whose weight is a lower bound for the weight of the optimal Hamiltonian cycle T^*_x. This bound is asymptotically tight, since only one edge has to be removed from T^*_{n+1} to make it a spanning tree (then, adding the minimum matching cannot increase the solution by more than $d(T^*_{n+1})/2$).

Now, suppose that the edge removed is the largest one incident to the new vertex. What one gets is an upper bound for $d(T_2)$ which decreases with $\max\{d(v_i, v_{n+1}), d(v_j, v_{n+1})\}$:

$$d(T_2) \leq \frac{3}{2} d(T^*_{n+1}) - \max\{d(v_i, v_{n+1}), d(v_j, v_{n+1})\} \tag{5.1}$$

where i and j are the neighbors of $n+1$ in T^*_{n+1}. On the other hand, the way T_1 is built clearly aims at minimizing the distance of the longest edge incident to $n+1$, so that this distance is at most $\max\{d(v_i, v_{n+1}), d(v_j, v_{n+1})\}$. By triangle inequality one gets:

$$d(T_1) \leq d(T^*_{n+1}) + 2\max\{d(v_i, v - n + 1), d(v_j, v_{n+1}).\} \tag{5.2}$$

Adding (5.1) and (5.2) with coefficients 2 and 1 gives the approximation ratio claimed. □

The method also applies to the insertion of k vertices, that tends to 3/2 for big values of k.

Theorem 5.11 ([8, 9])**.** *In the metric case,* MIN METRIC TSP$+_k$ *is approximable within ratio* $(3/2) - (1/(4k + 2))$*.*

Böckenhauer *et al.* [35] also studied the case where one single edge weight is modified. Recall that this problem is denoted MIN METRIC TSP $\mp$.

Theorem 5.12 ([35])**.** *In the metric case,* MIN METRIC TSP $\mp$ *is approximable within ratio* 7/5*.*

The proof of Theorem 5.12 is based on the observation that, if both the initial and the modified cost function are metric, the local change can only be moderate if the edge with changed cost is adjacent to an edge with small cost. The authors thus propose an algorithm which keeps the same tour when the local change is small, and compute a new solution when it is large. In this case, the approximation ratio is due to a high lower bound on all the distances of edges incident to the modified one, which derive from the metric hypothesis.

Asymmetric case. The MIN ASYMMETRIC TSP is another variant of the TSP that is of interest for many applications, as it generalizes the METRIC TSP. Unfortunately, this problem seems much harder to approximate METRIC TSP: while in the latter case a constant approximation is possible, for MIN ATSP the best known approximation ratio is $O(\log n)$ [57, 61].

Turning now to reoptimization, there exists a non-negligible gap between the approximability of the static version and that of the reoptimization version. In fact, Ausiello *et al.* [10] give constant approximation ratios for both MIN ATSP+ and MIN ATSP−, with very simple algorithms and proofs.

Theorem 5.13 ([10]). *In the metric case,* MIN ATSP+ *and* MIN ATSP− *are approximable within ratio* 2.

Proof. For MIN ATSP+, the algorithm simply inserts the new vertex arbitrarily in the initial optimal tour. By the metric hypothesis, none of the two new edges of the computed tour can exceed $d(T^*_{n+1})/2$. Considering that $d(T^*_n) \leq (T^*_{n+1})$, the result holds.

For MIN ATSP−, the algorithm skips the deleted node in the new tour, while visiting remaining nodes in the same order. Denote i and j the two neighbors of the deleted node $n+1$ in the initial optimal tour, that are reconnected in T:

$$d(T) = d(T^*_{n+1}) + d(i,j) - d(i,n+1) - d(n+1,j). \tag{5.3}$$

On the other hand, inserting $n+1$ in the new optimal tour between i and its neighbour say l, derives a feasible solution for the initial problem:

$$d(T^*_{n+1}) \leq d(T^*_n) + d(i,n+1) + d(n+1,l) + d(i,l).$$

Combining it with (5.3), some algebra leads to:

$$d(T) \leq d(T^*_n) + d(i,j) + d(j,i).$$

Thus, $d(T) \leq 2 \cdot d(T^*_n)$. □

5.5.2. MAX TSP

General case. While MIN TSP generally models vehicle routing and transportation problems, MAX TSP has a wide range of applications in DNA sequencing and data compression. MAX TSP is also well known to be NP-hard, but much easier than its minimization counterpart when it comes to approximation: it is approximable within a constant factor even when the distance matrix is completely arbitrary. In the static setting, the best known result for MAX TSP is a 0.6-approximation algorithm due to Kosaraju *et al.* [94].

Once again, the knowledge of an optimum solution to the initial instance seems to be very useful, since the reoptimization problem under insertion of a vertex can be approximated within a ratio of 0.66 (for large enough n) [10].

Theorem 5.14 ([10]). MAX TSP+ *is approximable within ratio* $(2/3) \cdot (1/n)$.

The basic idea of the algorithm in [10] is to output the best solution between T_1, computed with the *Best Insertion* procedure, and T_2, which results from the patching of a specific maximum cycle cover.

The cycle cover is computed so as to contain the two edges $(i, n+1)$,$(n+1, j)$ of the optimum (this is performed by an running the procedure with all possible pairs of vertices i, j).

The absolute error between T_1 and T^*_{n+1} is somewhat bounded by $a = d(i, n+1) + d(n+1, j)$. Indeed, i and j being the neighbors of $n+1$ in the new optimum T^*_{n+1}, removing them from T^*_{n+1} derives a Hamiltonian path on the initial set of nodes. This Hamiltonian path has a weight bounded by $d(T_1)$. On the other hand T_2 is produced out of a patching of cycles containing these two edges, so that its lower bound depends positively with a. Combining the bounds obtained on T_1 and T_2 leads to the claimed result.

Metric case. Even though MAX TSP is easier to approximate in the metric case than in the general case (the best approximation ratio to this day is 7/8 [69]), there is no polynomial time approximation scheme for this problem, meaning that there exists a constant c for which the problem is not c-approximable.

Ausellio *et al.* [8, 9] show that this bound can be broken in the reoptimization setting, under vertex insertion.

Theorem 5.15 ([8, 9]). MAX METRIC TSP+ *is approximable within ratio* $1 - O(n^{-1/2})$.

From this result, it is quite easy to show that the problem admits a polynomial time approximation scheme in this setting: if the desired approximation guarantee is $1 - \varepsilon$, for some $\varepsilon > 0$, just solve by enumeration the instances with $O(1/\epsilon^2)$ nodes, and use the result above for the instances with more nodes.

5.5.3. MIN LATENCY

Recall that MIN LATENCY aims at defining a tour T that minimizes the sum of latencies (instead of distances). The latency $l(i)$ of a node i amounts to the total distance between the node 1 and the node i along the tour T. The problem is much more complex than all the sub-problems we have surveyed so far. For example, in the special case when the metric is induced by a weighted tree, the MIN LATENCY remains NP-hard [118] while the METRIC TSP becomes trivial. The best static algorithm for MIN LATENCY gives 3.59 approximation-ratio [45].

One major problem when it comes to reoptimizing MIN LATENCY, is that a local modification might have a rather strong impact on the objective function, and thus deeply modify the structure of the solution. Nevertheless, a very easy 3-approximation algorithm can be devised under insertion of a new vertex.

Theorem 5.16 ([10]). MIN LATENCY+ *is approximable within ratio* 3.

The strategy achieving this result consists in inserting the new vertex in the last position. This is the only way not to modify the latency of any other vertex. Doing so, one only increases the overall latency by the latency of the new vertex meaning:

$$l(T) = l(T_n^*) + l(n+1) = l(T) + l(n) + d(n, n+1).$$

Considering that, $l(n) \leq l(T_n^*)$, and that due to the metric hypothesis, $d(n, n+1) \leq l(T_n^*)$, it is clear that the procedure can at most multiply by three the weight of the initial tree. By finally taking into account that $l(T_n^*) \leq l(T_{n+1}^*)$, the result follows.

6. Probabilistic combinatorial optimization

Given a combinatorial optimization problem Π, defined on a graph $G(V, E)$, its probabilistic counterpart PΠ is constructed by associating a system of probabilities with the instance G in the following way: each vertex $v_i \in V$ is provided with a probability p_i of being present in the instance that will actually need to be solved. In this sense, problem PΠ models the fact that the problem Π will have to be solved only on a sub-instance G', that is a sub-graph of G induced by an unknown subset of nodes $V' \subseteq V$. To illustrate the concept, we present three natural applications that can be found in the literature, and surveyed in [104].

The first application was presented in [80]. Consider a problem in which a break-down company wishes to optimize a tour across n potential clients. To this point, the problem amounts to the classical TRAVELING SALESMAN. Yet, the problem gets more complex when taking into account the fact that a client will not need break-down assistance everyday, so that, on a given day, only a subset of the n clients need to be visited in the tour. Moreover, the subset of clients that need assistance varies from one day to the next. Considering the frequency to which a given client v_i has needed assistance in the past, it is possible to determine the probability p_i for this client to need assistance on a random day. Associating its probability to each client, one gets an instance of PROBABILISTIC TRAVELING SALESMAN, initially presented and discussed in [80, 81]. One understands that the classical deterministic TSP somehow fails to grasp all aspects of such situations, and that a precise, operational, and rapidly applicable solution should be at the decision maker's disposal when confronted to such probabilistic problems.

The second application follows the work presented in [63], and deals with satellite shot planning. The problem is as follows: given a set of demands for photograph that can be processed by an orbiting satellite, one must decide which image will actually be taken by the satellite, and when. Consider that each demand is represented by a node, and that edges represent incompatibility relationships between these demands. Then, one has to find a maximum independent set on the graph so-obtained. Moreover, once a set of demands without bilateral incompatibility has been determined, one must determine a sequence of shots among them, on an oriented graph where arcs represent sequence relationship (some shots cannot be taken after or before others). In this graph, one has to determine a path with a maximum number of vertices. Although these models provide a rather rich and complex representation of the real-world system, they do not take a major parameter into account: meteorological conditions. Indeed, these conditions might prevent some shots from being taken, or at least they might lower there quality and precision. To take this parameter into account, it would suffice to associate a probability p_i with every demand v_i, p_i representing the probability for meteorological condition to be sufficiently good to actually take the shot v_i. Such a model would provide a more efficient tool to the decision maker, and could also provide some information on how robust a given solution is.

The third and last application deals with a timetable problem, where lessons must be assigned to classrooms, and can be found in [102, 103]. Quite classically, when lessons are modeled by vertices, with edges representing incompatibility relations between lessons (for example, because they are supposed to take place in the same timetable slot), the chromatic number of the so-obtained graph is the minimum number of classrooms needed to assign all lessons. Indeed, considering that each color represents a classroom, a coloring provides a feasible assignment of lessons to classrooms. Thus, in such a situation, the administration has to solve a MIN COLORING problem, or MIN k-COLORING problem (when the administration knows in advance that only k rooms will be available to assign all lessons). However administrations often have to define a timetable before the academic term begins, at a moment when it is still unclear whether some classes will actually take place or not, depending on the number of students who sign in. Associating with each vertex v_i its probability p_i to gather enough students for the class to take place, one gets a probabilistic version of the initial problem, much more adequate to the real-world situation.

These three applications show how central the notion of probability can be in some situations, and how PCOP's can manage to take this parameter into account.

6.1. Reoptimization and *a priori* optimization

Notice that up to now, we have only defined a probabilistic model for graph problems, where a vector of probabilities is associated with the set of vertices. It is now quite clear that such a model could be helpful in solving real-world problems. However we have said nothing on how to make use of such models, and how to take advantage of this additional information that these probabilities provide.

The first approach that comes to mind is to wait for the actual instance (*i.e.*, the sub-graph induced by the nodes actually present) to be revealed, and only then to compute a feasible solution on this particular instance, either optimal or approximate. This strategy, which will be referred to as *post-optimization* in what follows, does not take advantage of the probabilities, since the optimization process takes place after the Bernoulli trials have occurred. However, in many applications, the time window between the moment when the actual instance is revealed, and the moment when a solution on this instance is needed is too short for an optimum to be computed from scratch, especially when one deals with an NP-hard problem.

Another strategy, which could be referred to as *pre-optimization*, would consist in pre-computing all optima on all potential particular instances, and then only "pick" the optimum corresponding to the instance once it has been revealed. However, when all nodes are associated to probabilities in $]0,1[$ (so that no node is assured to be neither in nor out of the revealed instance), one understands that the number of potential instances goes up to 2^n. In such a case (which is actually the case that will be discussed), the pre-optimization strategy would require 2^n particular optima to be computed. The limit in computing resources clearly rules this strategy out in most cases.

For all these reasons, another approach has been developed. This strategy, called *a priori optimization* was introduced in [21, 80]. Basically, it consists in finding a solution on the initial instance, called *a priori solution*, and then to adapt this solution to the particular instance once it is revealed, through a very quick *modification strategy*.

Thus, this so-called modification strategy $\mathtt{M}$ should be a means to transform any feasible solution S on the initial instance $G(V, E)$, into a feasible solution denoted $S'(S, V', \mathtt{M})$, on any induced sub-graph $G[V']$, $V' \subseteq V$. In what follows, $\mathtt{M}$ will be an algorithm, which starting from $S[V']$ (the remaining part of the optimum), modifies it in order to obtain a feasible solution (optimal or approximate) on $G[V']$. We assume that $\mathtt{M}$ will not modify the solution when the revealed instance is actually the initial one, that is $S'(S, V, \mathtt{M}) = S$.

As we will see, not only is this method applicable under limited computational resources, but it also enables to model, evaluate, and optimize real-world strategies.

To illustrate this, let us go back to the break-down company example. Let us remind that, with a given probability, a client will not need break-down assistance on a particular day, so we can assume that only a subset of clients $V' \subseteq V$ will need to be visited. For various reasons, we can suppose that the company does not wish to reoptimize the tour everyday: either the gain associated with this strategy does not justify its computational cost, or the company might need a certain "stability" in its tours, for example by always visiting its clients in the same order.

To answer these various constraints, the following strategy can be proposed: a feasible tour through *all* clients can be computed once and for all. This tour S will be the so called *a priori* solution. Then, each day, it suffices to drop absent clients from the tour and only visit the present ones, following the order induced by the *a priori* tour S. This rather simple strategy clearly corresponds to actual behaviours when confronted with PCOP's, the modification strategy simply proposes an algorithmic formulation of this strategy. Following this idea, the choice of a modification strategy will be strongly related to the reality modeled. To illustrate this, consider the following example inspired from [24–26] and presented in [104].

The problem presented in [24–26] is PROBABILISTIC VEHICLE ROUTING. In its classical deterministic version, the goal is to determine a fixed set of routes of minimal total length through a set of n clients, which corresponds to the expected total length of the fixed set of routes plus the value of some extra travel distance that might be required. The extra distance will occur when the demand on one or more clients exceeds the capacity of a vehicle and causes it to go back to the depot before resuming its tour. Of course, in the probabilistic counterpart of this problem, a probability is associated with each client.

Considering this model, and an *a priori* solution, two distinct modification strategies might be implemented, corresponding to two distinct real-world situations:

- The vehicle goes through all clients following the *a priori* order, whether they require service during that particular problem instance or not, and it goes back to the depot as soon as it is full. This strategy will be denoted `MS1`;
- the vehicle follows the *a priori* order, but skips clients that do not require service. As before, it goes back to the depot as soon as it is full. This strategy will be denoted `MS2`.

Figure 1 provides an example of this model, and illustrates the two different modification strategies. Here, the *a priori* order is $\{0,1,2,3,4,5,6,0\}$ the node 0 is the depot, and nodes from 1 to 6 are associated with demands of 10, 20, 10, 20, 20 and 10, respectively. The vehicle's capacity is 30. In the figure, it is supposed that only nodes 2, 3 and 5 require service.

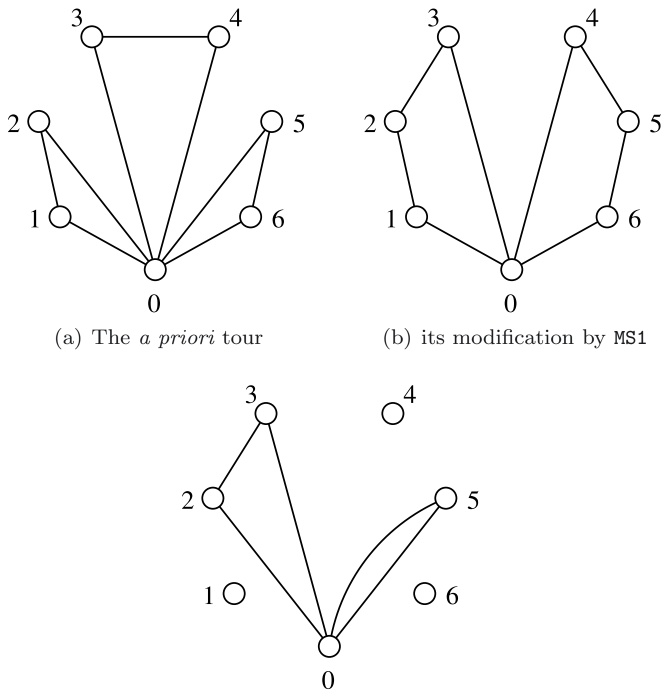
Considering the nature of the problem modeled here, it seems natural to assume that the graph is metric, so that a tour adapted by `MS2` will always be shorter than a tour adapted by `MS1`, due to the shortcuts it potentially takes with respect to the one adapted by `MS2`. However, something we wish to insist on, is that these two strategies are not "competitors", in so far as they allow to model two distinct realities. In that sense, both these strategies deserve a full analysis, despite the fact that one is intrinsically more efficient than the other.

`MS1` models the case where the information regarding demands are known only once the client is visited (so that the vehicle is bound to visit all clients anyway), whereas `MS2` models the case where all the information is available before the tour begins.

It is thus clear that the aim of probabilistic combinatorial optimization is *not* to look for the best modification strategy, but given a fixed modification strategy, chosen or designed accordingly to the reality modeled, to compute the best *a priori* solution. How one evaluates an *a priori* solution is what we are about to discuss.

6.2. Formalism and objective function

Let S be a feasible solution for a deterministic problem Π on a given graph $G(V,E)$, $\mathtt{M}$ be a modification strategy for Π and V' be a subset of V. Denote by $S'(S,V',\mathtt{M})$ the solution for Π in the induced sub-graph $G[V']$, obtained by applying the modification strategy $\mathtt{M}$ on the *a priori* solution S to make it feasible in $G[V']$. Denote $m(G[V'],S'(S,V',\mathtt{M}))$ its value. A reasonable requirement for $S'(S,V',\mathtt{M})$ is that $m(G[V'],S'(S,V',\mathtt{M}))$ is as close as possible to the value of



(a) The *a priori* tour (b) its modification by `MS1`

(c) and by `MS2`

FIGURE 1. Two different modification strategies.

an optimal solution for Π in $G[V']$, denoted by $\mathrm{opt}(G[V'])$. However, we cannot know *a priori*, which will be the sub-instance to be solved, since this instance results from n independent Bernoulli trials. Thus, taking advantage of the only information available regarding this instance, namely the probability vector, we will use as evaluation measure for S its *expectation*.

To compute its expectation, we need to determine the probability for each induced sub-graph to occur. Denote by $\Pr[V']$ the probability of presence of the vertices of V' only, hence the probability of the graph $G[V']$ to occur and set $\Pr(v_i) = p_i$, the presence-probability of vertex v_i. Then:

$$\Pr[V'] = \prod_{v_i \in V'} p_i \prod_{v_j \notin V'} (1 - p_j). \tag{6.1}$$

In particular, when all nodes are equiprobable ($\forall i,\ p_i = p$), which is a particular case commonly studied, (6.1) becomes:

$$\Pr[V'] = p^{|V'|}(1-p)^{n-|V'|}. \tag{6.2}$$

Considering that we have expressed the probability for each possible sub-graph to occur, as well as the value of each "modified" version of a given *a priori* solution, the expected value of a given *a priori* solution, which will be referred to as the *functional* in what follows, can be naturally expressed as:

$$E(G, S, \mathtt{M}) = \sum_{S' \subseteq S} \Pr[V'] m(G[V'],\ S'(S, V', \mathtt{M})) \tag{6.3}$$

where $\Pr[V']$ is as defined in (6.1).

As presented in [104], we will define formally a PCOP, within the *a priori* optimization framework, based upon the definition of an NP-optimization problem (NPO) given in Definition 4.1.

Definition 6.1 ([104]). Let Π be an NPO-problem defined as in Definition 4.1. A probabilistic version of Π, denoted by $\mathrm{P}\Pi$, is given by a sextuple: $((\mathcal{I}_\Pi, \mathbf{Pr}), Sol_\Pi, m_\Pi, goal(\Pi), \mathtt{M}, E_{\mathrm{P}\Pi})$ where:

- $\mathcal{I}_\Pi$ is as in Definition 4.1 and $\mathbf{Pr}$ is the set of all the vectors Pr of the presence probabilities of the data representing $I \in \mathcal{I}_\Pi$. The pair $(\mathcal{I}_\Pi, \mathbf{Pr})$ is the instance set of $\mathrm{P}\Pi$ and the couple $(I, \Pr[I])$, $I \in \mathcal{I}_\Pi$, $\Pr \in \mathbf{Pr}$, is an instance of $\mathrm{P}\Pi$;
- Sol_Π and $goal(\Pi)$ are as in the corresponding items of Definition 4.1;
- $\mathtt{M}$ is an algorithm, called modification strategy, such that, given an instance $(I, \Pr[I])$ of $\mathrm{P}\Pi$, a solution $S \in Sol(I, \Pr[I])$ and any sub-instance I' of I, it modifies S in order to produce a feasible solution $S'(S, I', \mathtt{M})$;
- $E_{\mathrm{P}\Pi}$ is the functional of S and is defined (analogously to (6.3)) as:

$$E_{\mathrm{P}\Pi}(G, S, \mathtt{M}) = \sum_{S' \subseteq S} \Pr[V'] m(G[V'],\ S'(S, V', \mathtt{M})) \tag{6.4}$$

  and $\Pr[V']$ is defined (analogously to (6.1)) as:

$$\Pr[V'] = \prod_{d_i \in I'} \Pr[d_i] \prod_{d_j \notin I'} (1 - \Pr[d_j])$$

  where d_i, d_j draw data of I and $\Pr[d_i]$ and $\Pr[d_j]$ their presence probabilities, respectively.

Notice that the modification strategy $\mathtt{M}$ is part of the definition of a given PCOP $\mathrm{P}\Pi$, so that two different modification strategies $\mathtt{M1}$ and $\mathtt{M2}$, applied to the same deterministic problem Π will produce two different PCOP's $\mathrm{P}\Pi1$ and $\mathrm{P}\Pi2$, which might have very different properties, in terms of complexity and approximability. This matches the remark we made earlier, namely that solving a PCOP does not consist in designing the best modification strategy, but it consists in computing the best *a priori* solution (meaning the *a priori* solution with optimal expected value), considering the modification strategy as a fixed parameter of the problem.

In practice, modification strategies are often very simple, and sometimes even trivial. Recall that these strategies are generally supposed to be algorithmic representations of real-life behaviours, so that they are required not to modify deeply the *a priori* solution. This requirement answers two different constraints: first, `M` should be a very fast algorithm (thus, it cannot afford to modify deeply the *a priori* solution), and second, for various reasons related to the situation modeled, a certain "stability" between the *a priori* and the modified solution can be desired. Revisit the example of the breakdown company. It can be required for various organizational or logistic reasons, that the clients are visited in the same order everyday, even if this order is far from the optimal one on some particular client subsets.

A very simple modification strategy used most frequently until now is the one consisting in dropping absent data out of the *a priori* solution and of taking the remaining elements of it as a solution for the realized instance. This strategy, which will be denoted by `MS` in the following, is feasible for numerous problems (this is the case of all the problems dealt in the monograph [104] as well as in [11, 12, 17, 21, 22, 27, 80–82, 84, 116]), but not for any problem. Indeed, some problems require the remaining elements to be completed with some additional elements to produce a feasible solution. For example, this is the case for the PROBABILISTIC MIN SPANNING TREE problem [39, 42].

Notice that *a priori* optimization under strategy `MS` amounts to the following robustness model for combinatorial optimization. Consider a generic instance I of a combinatorial optimization problem Π. Assume that Π is not to be necessarily solved on the whole I, but rather on a (unknown *a priori*) sub-instance $I' \subseteq I$. Suppose that each datum d_i in the data-set describing I is associated with a probability p_i, indicating how likely it is that $d_i \in I'$. Suppose finally that once the instance I' is revealed, the solver has no opportunity to solve the problem on this particular instance. Here, a simple and natural way to proceed is to compute an initial solution S for Π in the entire instance I and, once I' becomes known, to remove from S all elements of S that do not belong to I' (provided that this deletion results in a feasible solution for I') thus giving a solution feasible S' on I'. The objective is thus to determine an initial solution S for I such that, for any sub-instance $I' \subseteq I$ presented for optimization, the solution S' respects some given quality criterion (for example, optimal for I', or achieving constant approximation ratio, etc.).

Notice finally that a measure similar to (6.3) (and more generally to (6.4)) can be given for the post-optimization strategy. Denoting $S^*(V')$ the optimal solution on the sub-graph $G[V']$, and denoting $E^*(G)$ the expected value related to the post-optimization strategy:

$$E^*(G) = \sum_{S' \subseteq S} \Pr[V'] m(G[S'], S^*(V')) \tag{6.5}$$

This measure will be useful to bound approximation ratios. Indeed, recalling that this strategy computes the optimum for any sub-instance, its expected value constitutes an upper (resp., lower) bound for the expected value of the optimal *a priori* solution when dealing with maximization (resp., minimization) problems, regardless of the modification strategy.

6.3. Main methodological issues

6.3.1. Complexity issues

Membership in NPO is not always obvious. Following Definition 4.1, the objective function $m(S)$ associated with a NPO problem must be computable in polynomial time. Notice that, regarding PCOPs, this property is not always verified. Indeed, in PCOP's, the objective function is the expectation, and, following (6.4), the general formulation of this expectation consists in a sum with 2^n terms (one for each possible subset of I). Obviously, direct use of this general formula is not computable in polynomial time.

Thus, to prove that a given PCOP belongs to the NPO class, one must reformulate the functional in order to make it computable in polynomial time. In that sense, proving that a given PCOP belongs to the NPO class is one of the most essential issues, not only because the complexity result is interesting per se, but also because the proof involves a closed formulation of the functional, which is a key element when discussing the various aspects of the problem (characterization of the optimum, approximation, etc.).

Most of the studied PCOP's have been proved to be in NPO [11, 12, 17, 21, 22, 27, 42, 43, 80–82, 84, 99–101, 103, 116].

Of course, the reformulation of the functional is highly related to the modification strategy. In general, the most easy and natural way to perform this reformulation is to determine the probability for each datum of the instance to be present in the modified solution. To illustrate this, we will briefly present some results of [103], proving that PROBABILISTIC MAX INDEPENDENT SET-MS problem (associated with the modification strategy MS) is equivalent to MAX WEIGHTED INDEPENDENT SET.

Recall that the strategy MS consists in only removing absent data (here, vertices) from the solution. Thus, to be in the modified solution S', a given vertex v_i must be part of the *a priori* solution S, and must occur in the the graph $G[V']$, which happens with probability p_i. Finally, the probability for a given vertex v_i to be part of S' is:

- p_i if $v_i \in S$;
- 0 if $v_i \notin S$.

The expected number of vertices in S' (that is the functional we are trying to reformulate) is clearly the sum of probabilities of all nodes to be in S'. Thus:

$$E(G, S, \mathtt{MS}) = \sum_{v_i \in S} p_i. \tag{6.6}$$

First, notice that this expression is computable in polynomial time, so that PROBABILISTIC MAX INDEPENDENT SET-MS is in NPO. Notice also that maximizing this functional amounts to finding a maximum *weighted* independent set, in a graph where nodes are weighted by their respective probabilities, so that PROBABILISTIC MAX INDEPENDENT SET-MS and MAX WEIGHTED INDEPENDENT SET are equivalent problems, with respect to their computational complexities.

Finally, whenever the functional cannot be expressed by any polynomial expression, a statement about the complexity of its computation is impossible too. In this case, an interesting approach is to compute explicit (and non-trivial) bounds for it. This happens generally when the modification strategy is complex, or when it involves non-deterministic procedures.

For example, in [100], 5 different modification strategies are proposed and discussed for PROBABILISTIC MAX INDEPENDENT SET. The first modification strategy proposed is none other than the trivial strategy `MS`, which allows a polynomially computable expression for its associated functional. However, this strategy computes modified solutions which can be easily enhanced, since they often consist of non maximal stable sets. The four other modification strategies aim at rendering maximal modified solutions. However, none of them allows a closed expression of the functional. For example, the strategy denoted `U2` in the paper (defining the PCOP ΠIS2) consists in removing all absent nodes from the *a priori* solution, thus obtaining a -non maximal- stable set S', and to apply a greedy algorithm on the sub-graph induced by present nodes not having any neighbour in S'. Clearly, this modification strategy will always produce maximal stable sets, but it is impossible to determine the probability for a given node to be part of the modified solution. Yet, although the functional associated with `U2` cannot be expressed by any polynomial expression, it is proved the following.

Theorem 6.2 ([100]). *Approximation of an optimal a priori solution S^* by*

$$\bar{S} = \operatorname{argmax}\left\{\sum_{v_i \in \bar{S}} p_i : S \textit{ independent set of } G\right\}$$

guarantees for ΠIS2 approximation ratio:

$$\max\left\{\frac{p_{\min}}{1+p_{\max}}; \frac{1}{1+\Delta_G}\right\}.$$

Complexity of deterministic problems *vs.* **complexity of their probabilistic counterparts.** Note that PCOP's are generalizations of their deterministic counterparts. Indeed, any optimization problem Π can be considered as a particular case of PΠ, when all probabilities are equal to 1. This remains true regardless of the modification strategy associated with PΠ, since any modification strategy should not modify the *a priori* solution when all nodes are present.

This means that all negative results (hardness and inapproximability bounds) which apply to a problem Π remain valid for PΠ, unless P=NP. In particular, it is interesting to notice that if a given problem Π is NP-hard, and that its probabilistic counterpart has been proved to be in NPO (through reformulation of the functional, as we explained earlier), then one can immediately conclude that PΠ is also NP-hard. Conversely, when Π is polynomial, no immediate conclusion can be drawn regarding PΠ.

In this sense, two interesting issues have been discussed in the literature. The first one deals with the complexity of PΠ when Π is polynomial, and the second with the complexity of PΠ when Π is NP-hard, but in restricted class of instances in which Π can be solved in polynomial time.

Regarding the first issue, two polynomial problems were tackled in the probabilistic setting. In [82], the PROBABILISTIC SHORTEST PATH problem was introduced and discussed. In this setting, each node of a given graph $G(V, E)$ is associated with a presence probability, apart from two nodes s and t that are always present, and each edge is associated with a distance. The PROBABILISTIC SHORTEST PATH demands for a path with minimum expected length between s and t, when the modification strategy consists in skipping absent nodes in the a priori path. This problem is proved to be NP-hard, and some polynomial time algorithms are proposed for particular classes of graphs. In [23], another polynomial problem is proved to be NP-hard when considered in a probabilistic setting: the MIN SPANNING TREE problem. Here, the modification strategy consists in deleting absent nodes from the a priori tree, provided they do no disconnect the tree. Note that this version of PROBABILISTIC MIN SPANNING TREE is proved to be NP-hard, even when all edge weights are equal. In [39, 42], two other versions of PROBABILISTIC MIN SPANNING TREE (associated with two different modification strategies) are introduced and discussed. One of them is proved to be NP-hard, and the other one to be polynomial.

Regarding the second issue (polynomial restrictions of NP-hard problems), the same kind of results have been achieved. In particular, in [43], a trivial restriction of the MIN COLORING problem is tackled in the probabilistic framework. Indeed, it is proved that the PROBABILISTIC MINIMUM COLORING problem is NP-hard in bipartite graphs, although this class of graphs is trivially 2-colorable.

Finally, it is interesting to discuss restrictions regarding the probability vector, and their impact on the problem in terms of complexity and approximation. In particular, the case where all nodes are equiprobable ($p_i = p, \forall i$) is a commonly studied restriction, which often enables the definition of approximation algorithms, with approximation ratios bounded by expressions depending on p. Such results are provided for PROBABILISTIC TRAVELING SALESMAN in [116]: denote T^* an optimal *a priori* tour, and T^*_O an optimal deterministic tour a given graph G with equiprobable nodes. Then:

$$\frac{m(G, T^*)}{m(G, T^*_O)} \geq \frac{1}{p^2} \tag{6.7}$$

$$\frac{E(G, T^*_O, \mathtt{MS})}{E(G, T^*, \mathtt{MS})} \geq \frac{1}{p^2}\cdot \tag{6.8}$$

The bounds given in (6.7) and (6.8) show that a deterministic optimal tour constitutes a good approximation for the probabilistic one only in the case where p is large, *i.e.*, when the probabilistic version becomes "close" to the deterministic one. In particular, the approximation ratio reaches 1 (optimality) when $p = 1$.

6.3.2. Solution issues

In general, no direct link can be found between the *a priori* optimal solution and the deterministic optimal solution. In practice, the *a priori* optimum often constitutes a very poor solution for the deterministic problem, especially when the graph contains many nodes associated with low probabilities. For example, an optimal solution PROBABILISTIC MINIMUM COLORING might contain 3 colors or more in bipartite graphs.

To solve a given PCOP, it is absolutely necessary to characterize its optimum. Indeed, such a characterization is vital when designing efficient algorithms to compute optimal *a priori* solutions, and it is also a great help when leading approximation analyses.

Characterization of optimal a priori solutions. For numerous PCOP's, it is possible to characterize the optimal *a priori* solution with respect to parameters of the input graph. This is the case, in particular, when the modification strategy is very simple. Consider for example the case of PROBABILISTIC MAX INDEPENDENT SET associated with `MS` strategy, discussed in Section 6.3.1. Here, the optimal *a priori* solution can be clearly characterized as an optimal weighted independent set, in a graph where each nodes v_i is weighted by its probability p_i.

But such characterization is not always easy to define, and it gets harder and harder as the modification strategy gets more complex. This difficulty, and even sometimes impossibility to characterize clearly the optimum is often due to the fact that the weight of a given vertex (or edge) in the functional does not depend only on its own probability of occurrence, but also on the structure of the *a priori* solution. Put differently, this weight cannot be expressed as a function of the input graph.

Actually, even when using `MS` as modification strategy, it is still impossible to get a clear characterization of the optimal *a priori* solution for some problems. Consider for example the PROBABILISTIC LONGEST PATH problem, introduced in [99]. Under strategy `MS`, the functional of a given *a priori* solution $S = (0, 1, \dots k, k+1)$ (where $0, 1, \dots k, k+1$ are the vertices of the *a priori* path) is expressed as:

$$E(G, G, \mathtt{MS}) = \sum_{i=0}^{k} p_i p_{i+1} d(i, i+1) + \sum_{i=0}^{k-1} \sum_{j=i+2}^{k+1} p_i p_j \left(\prod_{l=i+1}^{j-1} (1 - p_l) \right) d(i, j) \quad (6.9)$$

where $d(i, j)$ is the weight (distance) of arc (i, j).

As one can see from (6.9), it is impossible to to express this probabilistic problem in terms of some weighted version of its deterministic support. The first term of the expression amounts to the weight of a path where each arc $(i, i+1)$ is associated with the weight $p_i p_{i+1} d(i, i+1)$. But the second term represents the impact on the functional of each potential "reconnecting" arc, integrated in S' by `MS` between vertices i and j when all vertices l $(l = i+1, \dots j-1)$ are absent from V', and both

i and j are present. Each of these edges might be part of the modified solution with probability $p_{ij}(S) = p_i p_j \left(\prod_{l=i+1}^{j-1}(1-p_l)\right)$, and thus their weight in the functional is $p_{ij}(S)d(i,j)$. Hence, the weight of an edge (i,j) $(0 \leq i \leq k-1, i+2 \leq j \leq k+1)$ is not only determined by characteristics attached to this edge (p_i, p_j and $d(i,j)$), but depends also on the probabilities of all vertices l, $i+1 \leq l \leq j-1$, and thus on the whole solution S. This is precisely why it is so hard to determine any property of the *a priori* optimum.

When the optimum cannot be characterized in any way, there is no possibility to design efficient optimal algorithms. Moreover, when dealing with approximation analyses, the only way to bound the expectation of the optimum is to bound it by the expectation associated with the post-optimization strategy, denoted E^* and defined in (6.5).

Polynomial sub-cases and stability. The identification of polynomial restrictive cases for NP-hard problems is always an interesting issue in complexity theory. This issue is also handled when tackling PCOP's, which are often NP-hard problems. The most common approach for such an issue is to start from polynomial instances for the deterministic version of the problem and to study if they remain polynomial for the probabilistic counterpart.

Consider, for example, the TRAVELING SALESMAN problem and its probabilistic version under MS. As it is shown in [20], matrices of the form $c_{ij} = a_i + b_j$ (called constant matrices) are the only ones where all the permutations of vertices have the same length, namely $\sum_{i=1}^{n} a_i + b_i$. Based upon this result, it is shown in [21] (see also [17]) that the constant matrices are the only ones that have the same expectation for any *a priori* tour T and this expectation is equal to $\sum_{i=1}^{n} p_i(a_i+b_i)$. Moreover, this fact remains true for any modification strategy. Thus, in the case of constant matrices, the probabilistic travelling salesman is polynomial, since it takes only $O(n)$ operations to build an arbitrary tour, that is optimal.

More generally, it is an interesting issue to identify class of graphs and probability vectors in which a deterministic optimum remains optimal in the probabilistic version of the problem, *i.e.*, identifying classes of instances for which $S^* = S_0^*$, where S_0^* and S^* are the optimal solutions of the deterministic problem and the optimal *a priori* solution of its probabilistic derivation (under some modification strategy M). Of course, in this case of stability between deterministic and *a priori* a optima, both versions of the problem are equivalent.

In the same line of ideas, another issue of interest is the study of conditions under which the *a priori* approach and the post-optimization one are equivalent, *i.e.*, identifying classes of instances for which $S'(G, V', \mathtt{M}) = S^*(V'), \forall V' \subseteq V$. This equivalence induces another kind of solution's stability in the sense that if, following M, we modify S_0^* to fit the present sub-instance $G[V']$ of the problem at hand, then the solution so obtained is optimal for this sub-instance. This interesting property is verified in the case of PROBABILISTIC TRAVELING SALESMAN with constant matrices, discussed earlier.

Polynomial approximation issues. As explained in [104], there exist four types of polynomial approximation results obtained for a probabilistic combinatorial optimization problem:

(1) one measures, for a given optimization problem, the quality of the *a priori* optimization with respect to the reoptimization; for some modification strategy $\mathtt{M}$, this can be done by means of the ratio $E(G,S,\mathtt{M})/E^*(G)$, where $E(G,S,\mathtt{M})$ and $E^*(G)$ are given by (6.3) and (6.5), respectively;
(2) one measures the quality of a solution S obtained in the (deterministic) support and without taking into account any probabilistic concept, when used as an *a priori* solution for the probabilistic counterpart (for a fixed modification strategy $\mathtt{M}$), this is done by means of the ratio $E(G,S,\mathtt{M})/E(G,S^*,\mathtt{M})$, where S^* is the optimal *a priori* solution (associated with $\mathtt{M}$) and both $E(G,S,\mathtt{M})$ and $E(G,S^*,\mathtt{M})$ are given by (6.3);
(3) one measures the quality of an *a priori* solution S, explicitly built for the probabilistic problem, this quality is measured by the ratio specified in item 2;
(4) finally, one can measure the expected approximation ratio achieved by the modification strategy $\mathtt{M}$, for a given *a priori* solution S, that is:

$$E\left(\frac{m(G[V'],S'(S,V',\mathtt{M}))}{\mathrm{opt}(G[V'])}\right)=\sum_{V'\subseteq V}\frac{m(G[V'],S'(S,V',\mathtt{M}))}{\mathrm{opt}(G[V'])}\Pr[V'].$$

For item 1, some results can be found in [83], which deals with the probabilistic travelling salesman and probabilistic minimum spanning tree, both problems defined on complete graphs with identical vertex probabilities and with vertices uniformly distributed on a plane, under strategy $\mathtt{MS}$.

For item 2, we quote the study performed in [21]. There, dealing with PROBABILISTIC TRAVELING SALESMAN under the same assumptions as in [83], the *a priori* tour T considered is the one computed by the celebrated Christofides' algorithm [46]. The *a priori* solution S considered is also frequently supposed to be a deterministic optimum, even when it is not computable in polynomial time. This is for example the case in [100], dealing with PROBABILISTIC MAX INDEPENDENT SET, the deterministic version of which does not admit any constant approximation algorithm.

Finally, for item 4, that constitutes a rather new measure in the framework of probabilistic optimization, even though it seems particularly adapted to this field, some results are provided in [39, 42], which deals with PROBABILISTIC MIN SPANNING TREE.

7. Final remarks

In Sections 3 to 5, we have presented main techniques and results achieved in the reoptimization paradigm. First, we have depicted a more general framework called *fully dynamic optimization* and related algorithms, which aim at maintaining optimal or approximate solutions under successive local perturbations of the instance. Then we have focused on the so-called reoptimization framework, where an initial optimal solution must be reoptimized after a *single* local perturbation of the initial instance. In this framework, many different new techniques were proposed. In particular, a whole array of problems, namely hereditary problems, can be easily approximated in this framework, since the initial optimum (or at least some parts of it when some of its elements are deleted) form a feasible solution on the perturbed instances. Reoptimization algorithm also achieve better approximation ratios than in the static setting, when applied to many other non-hereditary problems, in particular vehicle routing problems.

In Section 6, we have presented the notion of probabilistic combinatorial optimization problem, as well as the most studied way to deal with them: *a priori* optimization. Both these notions are very useful, because of their capacity to model real-world situations and behaviours. Tackling a PCOP through *a priori* optimization is quite a challenging task, that always results in many interesting issues to discuss, no matter how well-known, and even easy to solve, the deterministic support might be. Rather than reviewing the results achieved in this field, we have chosen to discuss these issues in detail, and to illustrate them with particular results from the literature when necessary. For a given problem PΠ these issues can be summed up as follows:

- reformulate the objective function, called functional, so as to prove that P$\Pi \in$ NPO. If such reformulation can not be achieved, on can try to provide non trivial bounds for its value;
- if $\Pi \in P$ (or similarly, if $\Pi \notin P$ but admits some polynomial sub-cases), discuss the complexity of PΠ, which might result in proving its NP-hardness (through a classical reduction proof), or polynomiality (through optimality of a polynomial algorithm);
- provide good characterization of optimal *a priori* solution;
- discuss polynomial sub-cases, and possibly stability of optima through the modification strategy `M`;
- provide approximation results.

Moreover, reminding that a given deterministic support results in different probabilistic versions, when associated with different modification strategy, one understands how rich a full analysis of a given PCOP can be.

It is quite clear that reoptimization and probabilistic optimization constitute complementary approaches when dealing with evolving instances.

Reoptimization handles cases where absolutely no information is available regarding the potential evolution of the instance, apart from knowing that it will remain local (*i.e.*, concern a constant number of elements). In this case, the goal is to design algorithms which take advantage of the initial optimum as much as possible, to compute efficiently good solutions on the perturbed one. However, it seems that regarding some specific perturbations, some other initial solutions could be adapted to the perturbed instance in a better way than the initial optimum.

On the other hand, probabilistic combinatorial optimization handles cases where some information is available regarding the potential perturbation (represented by probabilities), which by the way might more than local (*i.e.*, concern $O(n)$ elements), so that for a fixed adaptation strategy, one tries to design the initial solution which will be adapted in the best way in average.

One understands how deeply linked these two frameworks are and how, for a given problem, results achieved in one framework draw a new light on the analysis of the same problem in the other framework. For example, using a good reoptimization algorithm for a given problem (under vertex deletion) as adaptation strategy in the probabilistic version of the same problem, one could analyze to what extent the reoptimization version could provide better result when using specific kinds (and not necessarily optimal) solutions on the initial instance.

Acknowledgements. The useful comments of three anonymous referees are gratefully acknowledged. Many thanks to Cécile Murat and Jérôme Monnot for many helpful discussions.

References

[1] S. Albers, On randomized online scheduling, in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, ACM (2002) 134–143.

[2] S. Albers, Online algorithms: a survey. *Math. Program.* **97** (2003) 3–26.

[3] C. Archetti, L. Bertazzi and M.G. Speranza, Reoptimizing the traveling salesman problem. *Networks* **42** (2003) 154–159.

[4] C. Archetti, L. Bertazzi and M.G. Speranza, Reoptimizing the 0-1 knapsack problem. *Discrete Appl. Math.* **158** (2010) 1879–1887.

[5] T. Asano, K. Hori, T. Ono and T. Hirata, A theoretical framework of hybrid approaches to max sat, in *ISAAC*, *Lecture Notes in Computer Science* **1350**, edited by H.W. Leong, H. Imai and S. Jain. Springer (1997) 153–162.

[6] G. Ausiello, G.F. Italiano, A. Marchetti-Spaccamela and U. Nanni, Incremental algorithms for minimal length paths. *J. Algorithms* **12** (1991) 615–638.

[7] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie and M. Talamo, Algorithms for the on-line traveling salesman problem. *Algorithmica* **29** (2001) 560–581.

[8] G. Ausiello, B. Escoffier, J. Monnot and V.Th. Paschos, Reoptimization of minimum and maximum traveling salesman's tours, in *SWAT*, *Lecture Notes in Computer Science* **4059**, edited by L. Arge and R. Freivalds. Springer (2006) 196–207.

[9] G. Ausiello, B. Escoffier, J. Monnot and V.Th. Paschos, Reoptimization of minimum and maximum traveling salesman's tours. *J. Discrete Algorithms* **7** (2009) 453–463.

[10] G. Ausiello, V. Bonifaci and B. Escoffier, Complexity and approximation in reoptimization. *Cahier du LAMSADE* **281**. Université Paris-Dauphine (2008).

[11] I. Averbakh and O. Berman, Probabilistic sales-delivery man and sales-delivery facility location problems on a tree. *Transp. Sci.* **29** (1995) 184.

[12] I. Averbakh, O. Berman and D. Simchi-Levi, Probabilistic *a priori* routing-location problems. *Nav. Res. Logist.* **41** (1994) 973–989.

[13] R. Bar-Yehuda and S. Even, A linear-time approximation algorithm for the weighted vertex cover problem. *J. Algorithms* **2** (1981) 198–203.

[14] M. Bartusch, R.H. Möhring and F.J. Radermacher, A conceptional outline of a DSS for scheduling problems in the building industry. *Decis. Support Syst.* **5** (1989) 321–344.

[15] M. Bartusch, R.H. Möhring and F.J. Radermacher, Design aspects of an advanced model-oriented DSS for scheduling problems in civil engineering. *Decis. Support Syst.* **5** (1989) 321–344.

[16] J. Beardwood, J.H Halton and J.M Hammersley, The shortest path through many points, in *Mathematical Proceedings of the Cambridge Philosophical Society* **55**. Cambridge Univ Press (1959) 299–327.

[17] M. Bellalouna, *Problèmes d'optimisation combinatoires probabilistes.* Ph.D. thesis, École Nationale des Ponts et Chaussées, Paris, France (1993).

[18] M. Bellalouna, S. Souissi and B. Ycart, Average-Case Analysis for the Probabilistic Bin Packing Problem, in *Mathematics and Computer Science III: Algorithms, Trees, Combinatorics and Probabilities* (2004) 149–159.

[19] M. Bern and P. Plassmann, The Steiner problem with edge lengths 1 and 2. *Inf. Proc. Lett.* **32** (1989) 171–176.

[20] X. Berenguer, A characterization of linear admissible transformations for the m-travelling salesmen problem. *Eur. J. Oper. Res.* **3** (1979) 232–238.

[21] D.J. Bertsimas, *Probabilistic combinatorial optimization problems.* Ph.D. thesis, Massachusetts Institute of Technology (1988).

[22] D.J. Bertsimas, Traveling salesman facility location problems. *Transp. Sci.* **23** (1989) 184.

[23] D.J. Bertsimas, The probabilistic minimum spanning tree problem. *Networks* **20** (1990) 245–275.

[24] D.J. Bertsimas, A vehicle routing problem with stochastic demand. *Oper. Res.* **40** (1992) 574–585.

[25] D.J. Bertsimas and D. Simchi-Levi, A new generation of vehicle routing research: robust algorithms, addressing uncertainty. *Oper. Res.* **44** (1996) 286–304.

[26] D.J. Bertsimas, P. Jaillet and A.R. Odoni, *A priori* optimization. *Oper. Res.* **38** (1990) 1019–1033.

[27] D.J. Bertsimas, P. Jaillet and A.R. Odoni, *A priori* optimization. *Oper. Res.* **38** (1990) 1019–1033.

[28] D. Bilò, H.-J. Böckenhauer, J. Hromkovic, R. Královic, T. Mömke, P. Widmayer and A. Zych. Reoptimization of steiner trees, in *SWAT*, *Lecture Notes in Computer Science* **5124**, edited by J. Gudmundsson. Springer (2008) 258–269.

[29] D. Bilò, P. Widmayer and A. Zych, Reoptimization of weighted graph and covering problems, in *WAOA*, *Lecture Notes in Computer Science* **5426**, edited by E. Bampis and M. Skutella. Springer (2008) 201–213.

[30] D. Bilò, H.-J. Böckenhauer, D. Komm, R. Královic, T. Mömke, S. Seibert and A. Zych, Reoptimization of the shortest common superstring problem, in *CPM*, *Lecture Notes Computer Science* **5577**, edited by G. Kucherov and E. Ukkonen. Springer (2009) 78–91.

[31] M. Blom, S. Krumke, W. De Paepe and L. Stougie, The online-TSP against fair adversaries. *Algorithms and Complexity* (2000) 137–149.

[32] H.-J. Böckenhauer and D. Komm, Reoptimization of the metric deadline TSP. *J. Discrete Algorithms* **8** (2010) 87–100.

[33] H.-J. Böckenhauer, L. Forlizzi, J. Hromkovic, J. Kneis, J. Kupke, G. Proietti and P. Widmayer, Reusing optimal TSP solutions for locally modified input instances, in *IFIP TCS IFIP* **209**, edited by G. Navarro, L.E. Bertossi and Y. Kohayakawa. Springer (2006) 251–270.

[34] H.-J. Böckenhauer, L. Forlizzi, J. Hromkovic, J. Kneis, J. Kupke, G. Proietti and P. Widmayer, On the approximability of TSP on local modifications of optimally solved instances. *Algorithmic Operations Research* **2** (2007) 83–93.

[35] H.-J. Böckenhauer, J. Hromkovic, T. Mömke and P. Widmayer, On the hardness of reoptimization, in *SOFSEM*, *Lecture Notes in Computer Science* **4910**, edited by V. Geffert, J. Karhumäki, A. Bertoni, B. Preneel, P. Návrat and M. Bieliková. Springer (2008) 50–65.

[36] H.-J. Böckenhauer, J. Hromkovic, R. Královic, T. Mömke and P. Rossmanith, Reoptimization of steiner trees: Changing the terminal set. *Theor. Comput. Sci.* **410** (2009) 3428–3435.

[37] H.-J. Böckenhauer, K. Freiermuth, J. Hromkovic, T. Mömke, A. Sprock and B. Steffen, The steiner tree reoptimization problem with sharpened triangle inequality, in *CIAC*, *Lecture Notes in Computer Science* **6078**, edited by T. Calamoneri and J. Díaz. Springer (2010) 180–191.

[38] N. Boria and V.T. Paschos, Fast reoptimization for the minimum spanning tree problem. *J. Discrete Algorithms* **8** (2010) 296–310.

[39] N. Boria, C. Murat and V.T. Paschos, On the probabilistic MIN SPANNING TREE problem, in *IMCSIT* (2010) 893–900.

[40] N. Boria, J. Monnot and V. Th. Paschos, *Reoptimization of maximum weight induced hereditary subgraph problems*. Cahier du LAMSADE 311, LAMSADE, Université Paris-Dauphine (2011).

[41] N. Boria, J. Monnot and V. Th. Paschos, Reoptimization of the maximum weight P_k-FREE SUBGRAPH under vertex insertion, in *Proc. Workshop on Algorithms and Computation, WALCOM'12*, *Lect. Notes Comput. Sci.* Springer-Verlag (2011), to appear.

[42] N. Boria, C. Murat and V. Th. Paschos, On the PROBABILISTIC MIN SPANNING TREE problem. *J. Mathematical Modelling and Algorithms.* To appear.

[43] N. Bourgeois, F. Della Croce, B. Escoffier, C. Murat and V.Th. Paschos, Probabilistic graph-coloring in bipartite and split graphs. *J. Combin. Optim.* **17** (2009) 274–311.

[44] Z. Bouyahia, M. Bellalouna, P. Jaillet and K. Ghedira, *A priori* parallel machines scheduling. *Comput. Ind. Eng.* **58** (2010) 488–500.

[45] K. Chaudhuri, B. Godfrey, S. Rao and K. Talwar, Paths, trees, and minimum latency tours, in *FOCS*. IEEE Computer Society (2003) 36–45.

[46] N. Christofides, *Worst-case analysis of a new heuristic for the travelling salesman problem.* Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University (1976).

[47] M. Demange and V.T. Paschos, On-line vertex-covering. *Theor. Comput. Sci.* **332** (2005) 83–108.

[48] M. Demange and B. Leroy-Beaulieu, *Online coloring of comparability graphs: some results.* Report, Chair ROSE-2007-001, École Polytechnique Fédérale de Lausanne (2007).

[49] M. Demange, X. Paradon and V.Th. Paschos, On-line maximum-order induced hereditary subgraph problems, in *SOFSEM 2000 – Theory and Practice of Informatics*, *Lecture Notes in Computer Science* **1963**, edited by V. Hlaváč, K. G. Jeffery and J. Wiedermann. Springer-Verlag (2000) 326–334.

[50] M. Demange, X. Paradon and V.Th. Paschos, On-line maximum-order induced hereditary subgraph problems. *Int. Trans. Operat. Res.* **12** (2005) 185–201.

[51] M. Demange, G. Di Stefano and B. Leroy-Beaulieu, On the online track assignment problem. *Discrete Appl. Math.* To appear.

[52] C. Demetrescu and G.F. Italiano, A new approach to dynamic all pairs shortest paths. *J. ACM* **51** (2004) 968–992.

[53] M.L. Dertouzos and A.K. Mok, Multiprocessor online scheduling of hard-real-time tasks. *IEEE Trans. Softw. Eng.* **15** (2002) 1497–1506.

[54] D. Eppstein, Z. Galil, G.F. Italiano and A. Nissenzweig, Sparsification – a technique for speeding up dynamic graph algorithms. *J. ACM* **44** (1997) 669–696.

[55] B. Escoffier, M. Milanic and V.Th. Paschos, Simple and fast reoptimizations for the steiner tree problem. *Algorithmic Operations Research* **4** (2009) 86–94.

[56] S. Even and H. Gazit, Updating distances in dynamic graphs. *Methods Oper. Res.* **49** (1985) 371–387.

[57] U. Feige and M. Singh, Improved approximation ratios for traveling salesperson tours and paths in directed graphs, in *APPROX-RANDOM*, *Lecture Notes in Computer Science* **4627**, edited by M. Charikar, K. Jansen, O. Reingold and J.D.P. Rolim. Springer (2007) 104–118.

[58] G.N. Frederickson, Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput.* **14** (1985) 781–798.

[59] G.N. Frederickson, Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees. *SIAM J. Comput.* **26** (1997) 484–538.

[60] A.M. Frieze, On the value of a random minimum spanning tree problem. *Discrete Appl. Math.* **10** (1985) 47–56.

[61] A.M. Frieze, G. Galbiati and F. Maffioli, On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. *Networks* **12** (1982) 23–39.

[62] D. Frigioni, A. Marchetti-Spaccamela and U. Nanni, Fully dynamic algorithms for maintaining shortest paths trees. *J. Algorithms* **34** (2000) 251–281.

[63] V. Gabrel, A. Moulet, C. Murat and V.T. Paschos, A new single model and derived algorithms for the satellite shot planning problem using graph theory concepts. *A. Oper. Res.* **69** (1997) 115–134.

[64] J. Gallant, D. Maier and J.A. Storer, On finding minimal length superstrings. *J. Comput. Syst. Sci.* **20** (1980) 50–58.

[65] N.G. Hall and M.E. Posner, Sensitivity analysis for scheduling problems. *J. Scheduling* **7** (2004) 49–83.

[66] M.M. Halldórsson, Online coloring known graphs, in *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics (1999) 918.

[67] M.M. Halldórsson and J. Radhakrishnan, Greed is good: Approximating independent sets in sparse and bounded-degree graphs. *Algorithmica* **18** (1997) 145–163.

[68] M.M. Halldórsson, K. Iwama, S. Miyazaki and S. Taketomi, Online independent sets. *Theor. Comput. Sci.* **289** (2002) 953–962.

[69] R. Hassin and S. Rubinstein, A 7/8-approximation algorithm for metric Max TSP. *Inf. Proc. Lett.* **81** (2002) 247–251.

[70] M.R. Henzinger, Improved data structures for fully dynamic biconnectivity. *SIAM J. Comput.* **29** (2000) 1761–1815.

[71] M.R. Henzinger and V. King, Fully dynamic biconnectivity and transitive closure, in *FOCS*. IEEE Computer Society (1995) 664–672.

[72] M.R. Henzinger and V. King, Maintaining minimum spanning trees in dynamic graphs, in *ICALP*, *Lecture Notes in Computer Science* **1256**, edited by P. Degano, R. Gorrieri and A. Marchetti-Spaccamela. Springer (1997) 594–604.

[73] M.R. Henzinger and V. King, Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM* **46** (1999) 502–516.

[74] M.R. Henzinger and J.A. La Poutré, Certificates and fast algorithms for biconnectivity in fully-dynamic graphs, in *ESA*, *Lecture Notes in Computer Science* **979**, edited by P.G. Spirakis. Springer (1995) 171–184.

[75] J. Holm, K. de Lichtenberg and M. Thorup, Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM* **48** (2001) 723–760.

[76] L. Horchani and M. Bellalouna, The 2-dimensional probabilistic bin packing problem: an average case analysis of the FBS algorithm, in *Proceedings of the American Conference on Applied Mathematics*. World Scientific and Engineering Academy and Society (WSEAS) (2008) 449–453.

[77] O.H. Ibarra and C.E. Kim, Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM* **22** (1975) 463–468.

[78] Z. Ivković and E.L. Lloyd, Fully dynamic maintenance of vertex cover, in *Graph-Theoretic Concepts in Computer Science*. Springer (1994) 99–111.

[79] Z. Ivković and E.L. Lloyd, Fully dynamic algorithms for bin packing: Being (mostly) myopic helps. *SIAM J. Comput.* **28** (1998) 574–611.

[80] P. Jaillet, *Probabilistic traveling salesman problems.* Ph.D. thesis, Massachusetts Institute of Technology (1985).

[81] P. Jaillet, *A priori* solution of a traveling salesman problem in which a random subset of the customers are visited. *Oper. Res.* **36** (1988) 929–936.

[82] P. Jaillet, Shortest path problems with node failures. *Networks* **22** (1992) 589–605.

[83] P. Jaillet, Analysis of probabilistic combinatorial optimization problems in Euclidean spaces. *Math. Oper. Res.* **18** (1993) 51–70.

[84] P. Jaillet and A.R. Odoni, The probabilistic vehicle routing problem, in *Vehicle routing: Methods and Studies*, edited by B.L. Golden and A.A. Assad. North Holland, Amsterdam (1988) 293–318.

[85] P. Jaillet and M.R. Wagner, Online routing problems: Value of advanced information as improved competitive ratios. *Transp. Sci.* (2006) **40** 200–210.

[86] D.S. Johnson, *Near-optimal bin packing algorithms*. Ph.D. thesis, Massachusetts Institute of Technology (1973).

[87] D.S. Johnson, Fast algorithms for bin packing. *J. Comput. Syst. Sci.* **8** (1974) 272–314.

[88] D.S. Johnson and M.R. Garey, A 71/60 theorem for bin packing. *J. Complex.* **1** (1985) 65–106.

[89] N. Karmarkar and R.M. Karp, An efficient approximation scheme for the one-dimensional bin-packing problem, in *FOCS*, Chicago, IEEE Computer Society Illinois (1982) 312–320.

[90] R.M. Karp, Probabilistic analysis of partitioning algorithms for the traveling-salesman problem in the plane. *Math. Oper. Res.* **2** (1977) 209–224.

[91] R.M. Karp, A patching algorithm for the nonsymmetric traveling-salesman problem. *SIAM J. Comput.* **8** (1979) 561.

[92] V. King, Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs, in *FOCS*. IEEE Computer Society (1999) 81–91.

[93] P.N. Klein and S. Subramanian, A fully dynamic approximation scheme for shortest paths in planar graphs. *Algorithmica* **22** (1998) 235–249.

[94] S.R. Kosaraju, J.K. Park and C. Stein, Long tours and short superstrings (preliminary version), in *FOCS*. IEEE Computer Society (1994) 166–177.

[95] J.B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem, in *Proceedings of the American Mathematical Society* **7** (1956).

[96] P.S. Loubal, *A network evaluation procedure*. Bay Area Transportation Study Commission (1967).

[97] C. Lund and M. Yannakakis, The approximation of maximum subgraph problems, in *Proc. ICALP'93*, *Lecture Notes in Computer Science* **700**. edited by A. Lingas, R.G. Karlsson and S. Carlsson. Springer-Verlag (1993) 40–51.

[98] P.B. Miltersen, S. Subramanian, J.S. Vitter and R. Tamassia, Complexity models for incremental computation. *Theor. Comput. Sci.* **130** (1994) 203–236.

[99] C. Murat and V.Th. Paschos, The probabilistic longest path problem. *Networks* **33** (1999) 207–219.

[100] C. Murat and V.Th. Paschos, *A priori* optimization for the probabilistic maximum independent set problem. *Theor. Comput. Sci.* **270** (2002) 561–590.

[101] C. Murat and V.Th. Paschos, The probabilistic minimum vertex-covering problem. *Int. Trans. Oper. Res.* **9** (2002) 19–32.

[102] C. Murat and V. Paschos, The probabilistic minimum coloring problem, in *Graph-Theoretic Concepts in Computer Science*, Springer (2003) 346–357.

[103] C. Murat and V.T. Paschos, On the probabilistic minimum coloring and minimum k-coloring. *Discrete Appl. Math.* **154** (2006) 564–586.

[104] C. Murat and V.T. Paschos, *Probabilistic combinatorial optimization on graphs*. Wiley Online Library (2006).

[105] J. Murchland, *The effect of increasing or decreasing the length of a single arc on all shortest distances in a graph*. London Businness School, Transport Network Theory Unit (1967).

[106] C.H. Papadimitriou and M. Yannakakis, The traveling salesman problem with distances one and two. *Math. Oper. Res.* **18** (1993) 1–11.

[107] A. Paz and S. Moran, Non deterministic polynomial optimization problems and their approximations. *Theor. Comput. Sci.* **15** (1981) 251–277.

[108] K. Pruhs, E. Torng and J. Sgall, Online scheduling, in *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, edited by Joseph Y.-T. Leung, Chapter 15. CRC Press (2004) 15-1–15-41.

[109] M.B. Richey, Improved bounds for harmonic-based bin-packing algorithms. *Discrete Appl. Math.* **3** (1991) 203–227.
[110] N. Robertson and P.D. Seymour, Graph minors. XX. Wagner's conjecture. *J. Comb. Theory, Ser. B* **92** (2004) 325–357.
[111] G. Robins and A. Zelikovsky, Improved steiner tree approximation in graphs, in *SODA* (2000) 770–779.
[112] V.V. Rodionov, The parametric problem of shortest distances. *USSR Comput. Math. Math. Phys.* **8** (1968) 336–343.
[113] H. Rohnert, A dynamization of the all pairs least cost path problem, in *STACS*, *Lecture Notes in Computer Science* **182**, edited by K. Mehlhorn. Springer (1985) 279–286.
[114] S. Sahni and T.F. Gonzalez, P-complete approximation problems. *J. ACM* **23** (1976) 555–565.
[115] M.W. Schäffter, Scheduling with forbidden sets. *Discrete Appl. Math.* **72** (1997) 155–166.
[116] R. Seguin, Problèmes stochastiques de tournées de véhicules: un pas de plus vers le réalisme. *Cahiers du Centre d'études de recherche opérationnelle* **35** (1993) 187–226.
[117] J. Sgall, On-line scheduling-a survey, in *Online Algorithms: The State of the Art*, edited by A. Fiat and G.J. Woeginger, *Lect. Notes Comput. Sci.* **1442**. Springer (1998) 196–231. (1997).
[118] R. Sitters, The minimum latency problem is np-hard for weighted trees, in *IPCO*, *Lecture Notes in Computer Science* **2337**, edited by W. Cook and A.S. Schulz. Springer (2002) 230–239.
[119] D.D. Sleator and R.E. Tarjan, A data structure for dynamic trees. *J. Comput. Syst. Sci.* **26** (1983) 362–391.
[120] J.M. Steele, Subadditive Euclidean functionals and nonlinear growth in geometric probability. *Ann. Probab.* **9** (1981) 365–376.
[121] J.M. Steele, On Frieze's $\chi(3)$ limit for lengths of minimal spanning trees. *Discrete Appl. Math.* **18** (1987) 99–103.
[122] Z. Sweedyk, A 2+1/2-approximation algorithm for shortest superstring. *SIAM J. Comput.* **29** (1999) 954–986.
[123] A. Van Vliet, An improved lower bound for on-line bin-packing algorithms. *Inf. Proc. Lett.* **43** (1992) 277–284.
[124] V. Vassilevska, Explicit inapproximability bounds for the shortest superstring problem, in *MFCS*, *Lecture Notes in Computer Science* **3618**, edited by J. Jedrzejowicz and A. Szepietowski. Springer (2005) 793–800.
[125] A. Wagner, On finite affine line transitive planes. *Math. Zeitschr.* **87** (1965) 1–11.